2017

Financial Analysis with Artificial Neural Networks Short-term Stock Market Forecasting

Andrew Linzie
Gardner-Webb University

Financial Analysis with Artificial Neural Networks

Short-term Stock Market Forecasting

An Honors Thesis
Presented to
The University Honors Program
Gardner-Webb University
18 April 2017

by

Andrew Linzie

Accepted by the Honors Faculty

Dr. *Miroslaw Mystkowski*, Thesis Advisor	Dr. Tom Jones, Associate Dean, Univ. Honors
Dr. Eddie Stepp, Honors Committee	*Dr. Robert Bass, Honors Committee*
Dr. Candice Rome, Honors Committee	*Dr. Lorene Pagcaliwagan, Honors Committee*

Contents

1 Introduction

Seldom reward is absent from risk, and stock markets are a prime example. Stock markets across the world are viewed as profitable and risky at the same time. Companies have made a business out of forecasting these markets. Quantitative analysis companies use mathematicians, financial analysts, and computer scientists to compete in the stock market. The old days of floor trading have progressed towards high-frequency trading with supercomputers housed within the exchange. For example, the New York Stock exchange has created regulations for these companies so that there's competitive equality. The computer's power, length of cable to the exchange, and more has been standardized so that no single company will have an advantage with the exception to algorithms. Computers are delegated the buying and selling of stocks in the New York Stock exchange. A computer receives information from the market, decides an action in microseconds, and that decision gets sent to the exchange in milliseconds. From the computer's perspective, the difference between microseconds and millisecond is significant. The company's trading algorithms are secretive and protected, but their performance depends on time series analysis and machine learning theory.

Time series analysis is a popular method for forecasting financial systems, but over past decades, machine learning has become an essential area of research with relevant applications in classification and level estimation; both fit into the field of regression analysis within mathematics. Classification refers to the labeling of unseen data as a finite number of categories, while level estimation refers to guessing the numeric value of some process. For the stock market, classification refers to forecasting the direction of change, while predicting the price of the market is level estimation.

More generally, this thesis focuses on level estimation of blue chip stocks using artificial neural networks, a type of machine learning model, to forecast next-day closing values. Two different optimization methods are investigated to use with artificial neural networks. Back-propagation with stochastic gradient descent is the first method, and it is successful at forecasting nonlinear patterns in stock markets. The other model is an extreme learning machine, which is a method new to the recent decade. Models are trained, validated, and tested for measuring these two methods against each other to discover if extreme learning machines are comparatively successful.

Each chapter within this paper is purposed with informing the reader about the history, theory, and application of mathematics and computer science techniques for analyzing stocks. Chapter two presents the history of time series analysis, definitions, important concepts, and linear and nonlinear forecasting models. The structure and components of artificial neural networks are thoroughly explained within chapters three. Questions like, "*is the stock market predictable?"* or "*what does academic literature reflect about the success of artificial neural networks in financial markets?"* are answered within chapter four. Chapters five and six present the mathematical theory of stochastic gradient descent and extreme learning machines. Chapter seven explains the different methods which were used to receive the tabled results of chapter eight. Lastly, chapter nine interprets the results from chapter eight, chapter ten provides the thesis' conclusion, and the appendix contains all graphs and source code.

2 Time Series Analysis

2.1 Introduction

Time flows at a smooth pace and links together the phenomena of life, so quite naturally, the study of systems through time, time series analysis, is fundamentally important. A time series is an ordered set of observations by time and is plotted with time increasing to the right and the system's value increasing above. The purpose of studying a time series is to analyze the serial correlation of observations with future values, termed forecasting. Methods gleaned from time series analysis have been applied in many relevant fields like chemistry, biology, physics, psychology, finance, and business with the purpose of planning for the future.

Time series analysis is a branch of mathematics that investigates the dependence structure of a sample of observations. The application of theorems derived from time series analysis allows for accurate forecasting; the term accurate is user-defined, and the accuracy of these forecasts depend on numerous parameters. Some things to consider include the sample data's origin, amount of data, interval between observations, choice of model, estimation of model parameters, forecasting length, suspected noise, and desired accuracy are just a few of these important factors. The analyst has the freedom of choosing these parameters to receive their desired empirical results. With the uncertainty and freedom associated with time series analysis, one should question the goals of this branch.

The objective of studying time series includes basic application, theory, and model building. At the most basic level of analysis, one might desire to try to describe a system with the construction of a mathematical series. One might propose a hypothesis for explaining the behavior of a time series or relate the observation to imposing rules. Someone could use learned information from analyzing a time series to forecast into the future; forecasting relies on the assumption that future values will follow similar properties of the past. Determining the causation of a time series will give the analyst better forecasts. An analyst could use time series methods to return indicators of ominous events or empirically alter parameters to examine the results. The most general objective is to use the derived theory of time series for building predictive models. These models could be applied towards forecasting rain fall, the warming of the earth, or forecasting stock market values. Mathematics is the language of the cosmos, beautifully presented, structurally complex, and the applications of times series analysis are endless. [1]

2.2 History of Time Series Analysis

Time series analysis has been studied by mathematicians through history, but the theory of the field changed within the early 1900s. Within mathematics, researchers can become stuck on a problem, and approaching the problem from a different perspective is sometimes necessary for advancements. As mathematics has progressed through the centuries, the perspective of many branches changed, but time-series analysis resisted this change the longest.

The initial perspective of time-series analysis was deterministic as researchers believed and searched for the equations that described any time series without their errors. The errors within these systems were thought to be observation errors and not part of the underlying process. Analysts tried to predict change within financial markets with Fourier series and analogous methods. Trade, population, epidemics, and policy are a few of the many factors that affect financial markets, and these are not deterministic in any sense. [2] The study of differential equation is concerned with finding unique solutions to rates of change problems through time; many mathematicians have approached the study of time-series with differential equations. Until 1926, the search for the deterministic equations describing the cyclical movement of financial markets continued.

In 1927, Udny Yule, a British statistician, deviated from previous deterministic views by providing an analogy for a random component within an analyzed system. He started a wave

[1] Kendall, Maurice G. Time-Series. 2d ed. ed. New York :: Hafner Press, 1976, 12.
[2] Kendall, Maurice G. Time-Series. 2d ed. ed. New York :: Hafner Press, 1976, 4.

of new literature that contributing to the development of time-series analysis through the 20th century. Within his paper, “On a Method of Investigating Periodicities in Disturbed Series, with special reference to Wolfer’s Sunspot Numbers,” Yule describes the difference between *superposed fluctuations* and *true disturbances.* Here is an excerpt from the introduction of his paper explaining this idea:

> “When periodogram analysis is applied to data respecting any physical phenomenon in the expectation of eliciting one or more true periodicities, there is usually, as it seems to me, a tendency to start from the initial hypothesis that the periodicity or fluctuations are masked solely by such more or less random *superposed fluctuations* – fluctuations which do not in any way disturb the steady course of the underlying periodic function or functions. It is true that the periodogram itself will indicate the truth or otherwise of the hypothesis made, but there seems no reason for assuming it to be the hypothesis most likely *a priori.*
>
> If we observe at short equal intervals of time the departures of a simple harmonic pendulum from its position of rest, errors of observation will cause superposed fluctuations of the kind supposed in fig. 1. But by improvement of apparatus and automatic methods of recording, let us say, errors of observation are practically eliminated. The recording apparatus is left to itself, and unfortunately

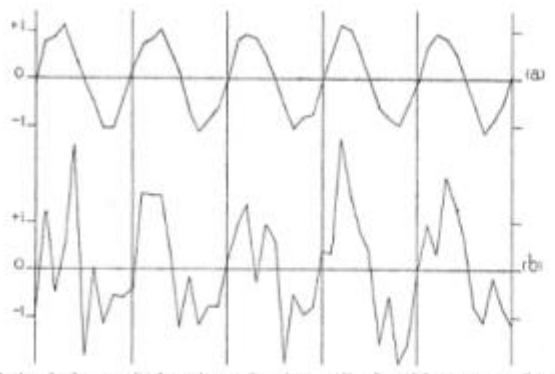


FIG. 1.—Graphs of simple harmonic functions of unit amplitude with superposed random fluctuations: (*a*) smaller fluctuations, (*b*) larger fluctuations.

Figure 2.1: Yule (1927)

boys get into the room and start pelting the pendulum with peas, sometimes from one side and sometimes from the other. The motion is now affected, not by *superposed fluctuations* but by true *disturbances,* and the effect on the graph will be of an entirely different kind. The graph will remain surprisingly smooth, but amplitude and phase will vary continually." [3]

Giving a better explanation, Yule is saying that there are random fluctuations that effect the underlying system. This idea that a system contains errors contributed to the idea that modeling time series should account for random fluctuations, modernly termed *shocks*. Today, these shocks are denoted by $\epsilon_t \sim N(0, \sigma^2)$, which is termed a random variable normally distributed with mean zero and variance sigma-squared. The concept of a process with its shocks leads to the concept of a *stochastic time-series* and is the subject of prolific discussion within academia. As people desire any predictive advantage within stock markets, the study of stochastic processes can be applied towards financial markets.

2.3 Time Series Definitions

2.3.1 Continuous and Discrete Time Series

A time series is a sequence of n observations ordered by time. Let $\{y_{t_n}\}$or $y_{t_1}, y_{t_2}, \dots, y_{t_n}$denote a time series at equally spaced intervals $t_{1,} t_{2,} \dots, t_n$for natural *n.* The length between these observations are different depending on the system being analyzed. For example, the time interval between measuring temperature and wind speed is much different than measuring the harvest of crops between years. Temperature and wind speed are examples of *continuous time series*, while the size of a harvest of crops between years is an example of a *discrete time series.* A continuous time series represents a constantly changing system over

[3] Yule, George Udny. "On a Method of Investigating Periodicities in Disturbed Series, with Special Reference to Wolfer's Sunspot Numbers." Philosophical Transactions of the Royal Society of London: Series A. Containing Papers of a Mathematical or Physical Character 226 (04/29/ 1927): 267-98. http://ezproxy.gardner-webb.edu/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=hsr&AN=521326766&site=eds-live.

an interval of time, while a discreet time series has a finite amount of observations. Within statistics, a time series has a mean and variance which are used for forecasting models.

A decomposition model may be used to better explain the interactions between different factors affecting the change in a series. Let, y_t be the value, s_t be the seasonal variation, r_t be the trend, $c_t$be the cyclic component, and let $\epsilon_t \sim N(0, \sigma^2)$ be the irregular component of the series, then the decomposition model can be presented as:

$$y_t = s_t + r_t + c_t + \epsilon_t$$

The seasonal variation is the pattern influenced by the time of the year, the trend of a series reflects the long-term progression, the cyclic component is the repeating but non-periodic pattern present within a time series, and there is always a shock value ϵ_t in an observable process. This is to say that many factors go into the value of a time series, which effects the visual randomness. The different components of y_t are recognizable in international airline passenger data, which measures the number of people traveling internationally each year. The seasonal variation, trend, cycle, and shock are obvious.

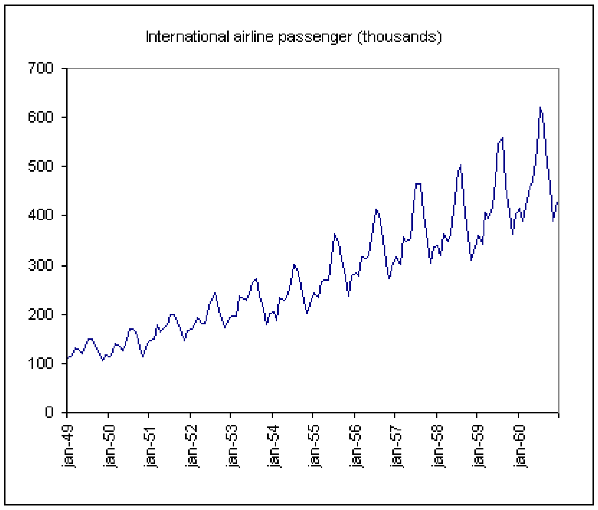


Figure 2.2:Airline Passenger Time Series

2.3.2 Stationary versus Non-Stationary

The study of time series forecasting is divided into two types of stochastic models termed *stationary* and *non-stationary* models. Stationary models assume that the process remains within a statistical equilibrium where the mean and variance of the time series do not vary over time; a non-stationary process has no constant mean and variance level. If a trend exists within the data, then the process is non-stationary. Stationarity is an unrealistic quality to assume for an industry, business, or economically related process, but stationary models have been shown to be appropriate for some non-stationary processes. [4] Non-stationary time series require different forecasting methodology because they possess greater variation.

2.3.3 Noisy Processes

As Yule suggested the noise of the surrounding environment combine into a third series. Some sources of noise come from political, technological, or geographic events within the world, and this noise is reflected in stock prices. In particular, the stock market is assumed to be a non-stationary process, forecasting stocks is modeled with non-stationary methods.

There are numerous examples of external noise which can complicate mathematical model building. Within the first few days of his term, President Donald Trump claimed on Twitter that the Federal government would not build a fleet of Air Force One planes. Ignoring political opinions, Boeing's stock took an immediate dive following those comments because those planes represented over $4 billion in contracts, which Boeing had begun building.

2.4 Linear and Nonlinear Forecasting Models

2.4.1 Introduction

The purpose of this section is to describe common methods used for general forecasting. Linear and nonlinear forecasting models, the ideas of model selection, and parameter estimation are presented. The common idea of the models presented below is that they rely on already known information to forecast future values of the process.

Linear models are appropriate for short-term forecasting, they have the advantage of being very quick, but they lack the accuracy and adaptive qualities that non-linear models possess. If the market has a threshold response where the price hits a certain level and starts to climb rapidly, a linear model would not react quickly to the change or be able to predict the

[4] Box, George E. P., Gwilym M. Jenkins, Gregory C. Reinsel, and Greta M. Ljung. Time Series Analysis : Forecasting and Control. Fifth edition / ed. Hoboken, New Jersey :: John Wiley & Sons, Inc., 2016. 1 online resource, 1-10.

end of the trend. Similarly, if there were a slow climb in price to a sudden drop, termed a bubble, then a linear model would not perform well in this situation either. To summarize, linear models are not accurate for highly volatile markets where the changes in price are sudden and large, but they offer a useful perspective which helps to understand the method of forecasting.

2.4.2 Linear Regression

The simplest model that everyone learns within statistics or economic classes is the linear regression model, which provides a best-fit line to a sample of data. The deterministic equation modeling linear regression is given below.

$$y_t = \sum \beta_k x_{k,t} + \epsilon_t$$

Within this model, the future values y_t are based upon the linear combination of the input values $x_{k,t}$ with its estimated β_k value and the disturbance term ϵ_t. The parameters needed to use the linear regression method requires $\widehat{\beta_k}$, an estimation of β, and this is estimated by minimizing the sum of square differences between the actual observation y_t and the observations predicted by the linear model, $\widehat{y_t}$. [5] Less obviously, McNelis proposes the estimation problem as

$$Min\Psi = \sum_{t=1}^{T} \hat{\epsilon}_t^2 = \sum_{t=1}^{T} (y_t - \widehat{y_t})^2$$

$$s.t.\ y_t = \sum \beta_k x_{k,t} + \epsilon_t$$

$$\widehat{y_t} = \sum \hat{\beta} x_{k,t}$$

[5] McNelis, Paul D. Neural Networks in Finance : Gaining Predictive Edge in the Market. Academic Press Advanced Finance Series. Burlington, MA: Academic Press, 2005, 14-17.

$$\epsilon_t \sim N(0, \sigma^2)$$

2.4.3 Generalized Autoregressive Conditional Heteroskedasticity

With nonlinear models, people try to approximate the underlying process by creating non-linear functional forms and create assumptions for estimating parameters. Proposed by Bollerslev(1986, 1987) and Engle (1982), GARCH is a method for forecasting, and Engle received the Nobel Prize in 2003 for his research on this model. [6,7,8] The $GARCH(p,q)$ model has σ^2 terms of order p and the other components is a $ARCH(q)$ model.

First, this model involves forecasting the variance of a process based upon previous variance forecasts and parameters, which defines the evolution of the conditional variance. McNelis thoroughly explains, "the variance of the disturbance term directly affects the mean of the dependent variable and evolves through time as a function of its own past value and the past squared prediction error." [5] Below, a mathematical description of the GARCH model is provided.

$$y_t = x_t' b + \epsilon_t$$

$$\epsilon_t | \psi_t \sim N(0, \sigma_t^2)$$

$$\sigma_t^2 = \omega + \alpha_1 \epsilon_{t-1}^2 + \cdots + \alpha_q \epsilon_{t-q}^2 + \beta_1 \sigma_{t-1}^2 + \cdots + \beta_p \sigma_{t-p}^2 = \omega + \sum_{i=1}^{q} \alpha_i \epsilon_{t-i}^2 + \sum_{i=1}^{p} \beta_i \sigma_{t-i}^2$$

The parameters for the different variables can be found by the typical method of maximizing the sum of the logarithm likelihood function over the entire sample.

[6] Bollerslev, Tim. "Generalized Autoregressive Conditional Heteroskedasticity." Journal of Econometrics 31, no. 3 (1986): 307-27. http://ezproxy.gardner-webb.edu/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=bth&AN=4987693&site=eds-live.
[7] Bollerslev, Tim. "A Conditionally Heteroskedastic Time Series Model for Speculative Prices and Rates of Return." 1987.
[8] Engle, Robert F. "Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflation." Econometrica 50 (07// 1982): 987-1007. http://ezproxy.gardner-webb.edu/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=hsr&AN=521173546&site=eds-live.

For those not familiar with statistics, the main point of this chapters was to stress the mathematical logic of forecasting; the future is estimated with known or historical values. Forecasting with machine learning is similar because an artificial neural network is trained on previous values to forecast stocks. There are many other models that could have been presented within this section, but there's nothing to gain from presenting this information since all future information relates to machine learning.

3 Artificial Neural Networks

3.1 Introduction

The previous chapter about time series analysis discussed the classical techniques of estimating the future values of some process. The process is assumed to be stochastic, and the process may be stationary or non-stationary. These properties alone are not an exhaustible list of the ways to classify a system, but they are the most important when determining which type of model to use. The question now becomes, *is this process better estimated with a linear or a non-linear model?* Since this thesis is concerned with the forecasting stock markets, a non-linear model is necessary for regression. As discussed in previous chapters, the noise from geopolitical events can drastically effect the accuracy of forecast. The changes within the stock market can be violent and sudden, and these changes are most accurately predicted with a non-linear model.

The pure time series analysis approach includes building a model, parameter estimating, and determining how to update those parameters to fit the future. Machine learning is the other method of approach, and machine learning has become popular for forecasting in recent years. The machine learning approach is significantly more complicated because the method requires the user to know how to program, build models, estimate parameters, and structure data to be successful. From academic articles detailed within the review of literature chapter, the consensus among researchers is that machine learning models, when correctly tuned, always outperform time series analysis models. The machine learning approach used within this thesis is known as an artificial neural network.

Artificial neural networks are used on a regular basis by people across the world. Hand writing classification at the post office, speech synthesis from Google or Siri, and image classification are all examples of regression with neural networks. They can be applied to nearly any problem that has available data and can be quantified numerically or by labels. The method of training, validating, and testing neural networks is adopted within this thesis for the purpose of stock market forecasting. A detailed description is provided within this chapter for understanding the mathematical theory presented in chapters five and six. Artificial neural networks have a detailed history, and for those who are interested in the origins of neural networks, Warren McCulloch, Walter Pitts, Donald Hebb, Frank Rosenblatt, David E. Rumelhart, James McClelland, and many more are the innovators of artificial neural networks. [9,10,11,12]

3.2 Layers

Artificial neural networks were developed with inspiration from the human brain's structure. The human brain contains neurons, synapses, and electrical signals communicated over those synapses. An artificial neural network has similar components with nodes,

[9] Cowan, Jack. "Discussion: Mcculloch-Pitts and Related Neural Nets from 1943 to 1989." Bulletin of Mathematical Biology 52, no. 1/2 (01// 1990): 73. http://ezproxy.gardner-webb.edu/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=edb&AN=72259295&site=eds-live.
[10] Hebb, D. O. The Organization of Behavior : A Neuropsychological Theory. Wiley's Books in Clinical Psychology: New York : Wiley, 1949., 1949.
[11] Rosenblatt, F. "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain." In Neurocomputing: Foundations of Research., edited by James A. Anderson and Edward Rosenfeld, 92-114. Cambridge, MA, US: The MIT Press, 1988.
[12] Rumelhart, D. E. and et al. "Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations." Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations (1986): NoPg. http://ezproxy.gardner-webb.edu/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=20h&AN=33837881&site=eds-live.

connections, and weights representing the human brain's neurons, synapses, and strengths of electrical signals, respectively. Humans are able to logically deduce, artistically express themselves, and possess consciousness because of the brain's structure. An artificial neural network implements that structure, and there are three different layers within a neural network: the input layer, the hidden layer, and the output layer. An artificial neural network can take on many forms where connections, loops, and multiple directions of connections are permitted, but only one type of neural network is presented within this paper. Do not be mistaken, while they are similar, artificial neural networks do not resemble humans brains in size or abstract reasoning. Google is working on creating an human brain sized artificial neural network; Although, empirical observation has proven that neural networks do not need extensive computational power for successful domain specific regression. Feed-forward neural networks are a subset of the many types of neural networks, and they are explored below.

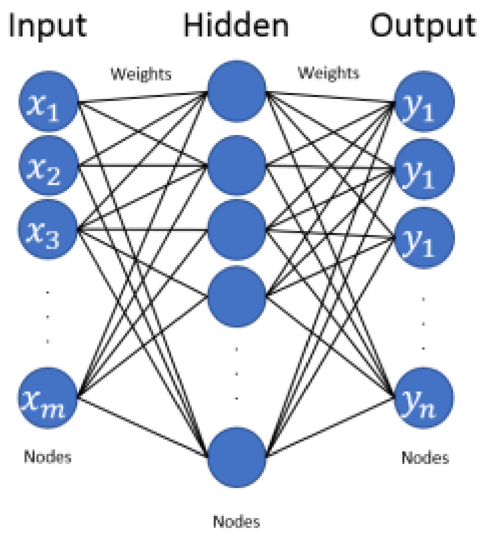


Figure 3.1: A general artificial neural network architecture

A feed forward neural network is like the function $y = f(x)$ in mathematics. The input layer is where information is "fed" into the model to provide numbers for the hidden layer to mathematically transform. The hidden layer uses an activation function to squash the inputs for the output layer. Depending on the type of problem a neural network is build for, many hidden layers may be needed to receive accurate results. A neural network with only one hidden layer is termed a "Single-layer feed-forward neural network," and single-layer feed forward neural networks are the only type used within this thesis. The output layer collects the results of regression or classification within the hidden layer and sums the results. Between these three layers, there are connections from every node to every other node within the next layer, and on each of these connections there is an associated number which determines the strength of the connection. The specifics of these components are explained in detail in parts 4.3, 4.4, and 4.5.

3.3 Nodes

The nodes of a neural network resemble the neurons of a human brain, and the nodes within each of the three layers have a different function. The nodes within the input layer are purposed with mapping an input vector x multiplied by weights into the hidden nodes. The hidden layer nodes take a vector $x * w$ from the previous layer for inputting into an activation function. The output layer nodes sum the weights multiplied by the hidden layer node outputs, and the results are the output of each output node. Mathematically, this process can be expressed as the output at the k^{th} node.

$$y_k = \phi(\sum_{j=0}^{m} w_{kj} x_j)$$

This formula explains the process of propagating values from the input nodes towards the output nodes; the previous layer node's outputs are multiplied by their weights, summed, and the summed value is the input to an activation function. To better understand this process, refer to figure 4.

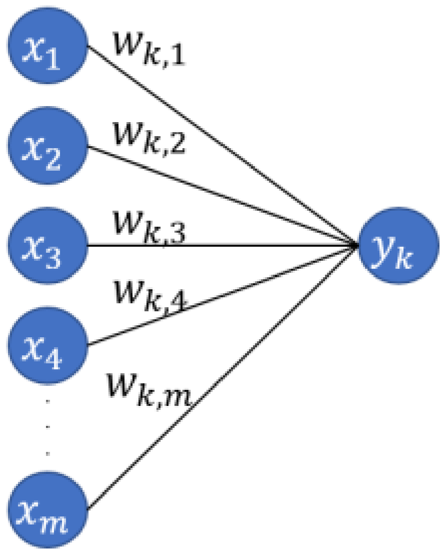


Figure 3.2: Perceptron

The number of nodes contained within each layer is dependent on the domain specific application. The number of nodes within the input layer is the same number of features a researcher thinks accurately represents the amount of information which should be provided to the neural network to receive correct classifications within the output layer. The number of hidden layer nodes is empirically determined through the process of model selection and varies; Given n input nodes, the Kolmogorov-Arnold Representation Theorem gives the general rule that $2n + 1$ nodes with a single hidden-layer neural network can approximate any nonlinear piecewise continuous function. [13] The output layer contains the same amount of nodes as the target values of the neural network.

Applying this knowledge to the stock market, the input nodes or features, would be relevant information like the close, open, volume, high, low, technical indicators, or other information deemed important for regression. The hidden nodes would be chosen by the method of iteratively determining the best number or by using a validation set. The output layer typically consists of only one node when applied to forecasting stock markets. The level estimation method is used to predict the next closing value t time units into the future, and the classification method is used for forecasting the sign change of a stock t time units into the future.

3.4 Weights

The weights of the neural network are initialized randomly and modified to solve the transformation problem between the input and output layers. Within a single-layer feed-forward neural network, there are two sets of weights; the first set of weights exists between the input and hidden layers while the other set is between the hidden and output layers. Each node has a connection and corresponding weight to every other node in the next layer. The purpose of this connection scheme is to control how influential each feature is in reducing the

[13] Tikhomirov, V. M. "On the Representation of Continuous Functions of Several Variables as Superpositions of Continuous Functions of a Smaller Number of Variables." In Selected Works of A. N. Kolmogorov: Volume I: Mathematics and Mechanics, edited by V. M. Tikhomirov, 378-82. Dordrecht: Springer Netherlands, 1991.

error within output layer. Any weights within the neural network are initialized within the interval $[0, 1]$. A weight with a value of 1 allows the previous layer node to be "fully on," while a weight with the value of 0 turn that node "fully off." Along with the activation function, number of nodes within different layers, and available data sets, the weights are one of the most important component within a neural network. The weights of a neural network transform an input from an n-dimension space into an output in an m-dimension space, which is mathematically described as $f: \mathbb{R}^m \rightarrow \mathbb{R}^n$.

3.5 The Activation Function

The activation function of a neural network determines the output of a node. It is termed an activation function because the value outputted at each node effects or activates the next layer of nodes. An activation function is also called a "squasher function" because it limits the output of each node to a predetermined range. The activation function is meant to model the process of natural learning within the human brain. An important characteristic of the function is that a small change in the input to the node has a corresponding small change in the output of the node. It is necessary for an activation function to be differentiable at all points, which is discussed further in chapter five. [14]

The input layer uses no activation function, while the hidden layer always uses an activation function. The output layer can use an activation function but typically uses the linear activation function $f(x) = x$. The output layer is used to sum the results of the transformation within the hidden layer, and does not need to modify the classified data. There are many different types of activation functions which may be used for neural networks. Examples of continuously differentiable nonlinear activation functions commonly used in neural networks

[14] Haykin, Simon. Neural Networks: A Comprehensive Foundation. Prentice Hall PTR, 1998.

are sigmoidal nonlinearities, which includes the logistic function and the hyperbolic tangent

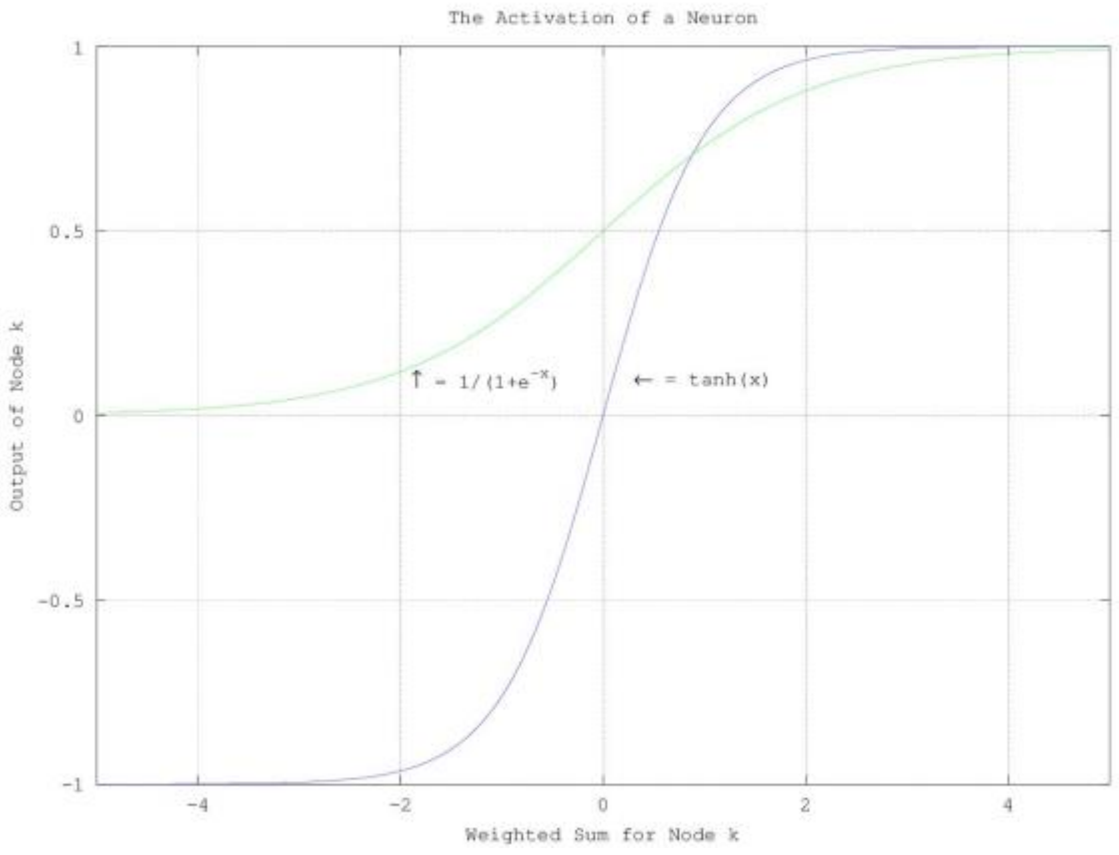


Figure 3.3: Logistic and Hyperbolic Tangent Functions

function (refer to figure 5).

It is important to include a few observations about the domain and range of the logistic and hyperbolic tangent function. As their inputs approach infinity, the limit of the logistic function approaches 0 and 1 while hyperbolic tangent approaches -1 and 1. The range changes very little for both function where they asymptotically approach their limits, and finding weights which minimize error for large inputs is difficult. To remedy this problem, data is scaled to the domain of $[0, 1]$ or $[-1, 1]$ prior to training, and the choice of the interval depends on the method of scaling and activation function. The effect of scaling input data to a specified range allows the network to fully take advantage of the activation function. This step is known as *preprocessing,* and the practical implementation is discussed in chapter seven.

3.6 Performance of a Neural Network

The main point and take-away stressed within this chapter is the possibility of taking some input in an n-dimensional space and transforming it into an m-dimensional space, which means that neural networks can be used for classifying data. Whether this method is successful or not is another question entirely. The success of a neural network depends on many different factors relating to its structure, provided information, and optimization method.

In practice, the choosing the structure of the network stems from theory or past researcher's success. Tuning the network for the domain specific problems involves experimenting with choosing different parameters iteratively or exhaustively and influences the success of a neural network. Within chapters five and six, two methods are proposed for updating the weights of a neural network to successfully forecast the next day's closing price, and chapter seven highlights the implementation of those optimization methods.

4 Review of Literature

The results and ideas presented within this paper would not be possible without the contributions from researchers of the past, so this chapter presents previous research on market analysis. The concepts of time series analysis and artificial neural networks have been detailed, but the advantages and disadvantages of either one has not been discussed. Many questions arise when forecasting short-term stock returns like *is the stock market random or predictable? Do time series analysis methods outperform machine learning methods? Which neural network weight optimization method creates better forecasts?* These questions are answered through the work of other researchers, and all decisions made in chapter seven have been based upon the successes and failure of others. When discussing financial forecasting, there are three beliefs which researchers commonly hold towards the methods and results of for profit market analysis.

One school of thought asserts that no one can achieve better than average accuracy in predicting the direction or level of change in the stock market. Fama created the Efficient Market Hypothesis to reflect this idea, and Jensen believes that there is concrete proof that the Efficient Market Hypothesis holds. [15,16] The Efficient Market Hypothesis states that all markets are extremely efficient, meaning all the information of a stock is already reflected in its price; it's impossible to profit from signal analysis and model building similar to the methods presented within chapter two and three. This belief was widely debated within the financial community in the 1970s, but many academic articles have found results contradicting this theory. The Efficient Market Hypothesis is commonly used to argue that financial markets follow a Random Walk Model; this is called the Random Walk Hypothesis.

Lo and MacKinlay tested the random walk hypothesis for weekly stock market returns by comparing variance estimators derived from data sampled at different frequencies. They concluded that the random walk model was strongly rejected for the entire sample period and for all sub-periods for a variety of aggregate return indexes and size-sorted portfolios.[17] Similarly, Lendasse, et. al., had the goal of breaking the Random Walk Hypothesis with non-linear statistical methods, and they were able to create an artificial neural network model which found non-linear relationships within the stock data.[18] Hassan, Nath, and Kirley used Hybrid Hidden Markov, artificial neural network, and genetic algorithm models with the goal of

[15] Fama, Eugene F. "Efficient Capital Markets: A Review of Theory and Empirical Work." The Journal of Finance 25, no. 2 (1970): 383-417. http://dx.doi.org/10.2307/2325486.
[16] Jensen, Michael C. "Some Anomalous Evidence Regarding Market Efficiency." Journal of Financial Economics 6, no. 2 (1978/06/01 1978): 95-101. http://dx.doi.org/http://dx.doi.org/10.1016/0304-405X(78)90025-9.
[17] Lo, Andrew W. and A. Craig MacKinlay. "Stock Market Prices Do Not Follow Random Walks: Evidence from a Simple Specification Test." The Review of Financial Studies 1, no. 1 (1988): 41-66. http://dx.doi.org/10.1093/rfs/1.1.41.
[18] Lendasse, A., E. de Bodt, V. Wertz, and M. Verleysen. "Non-Linear Financial Time Series Forecasting - Application to the Bel 20 Stock Market Index." European Journal of Economic and Social Systems 14, no. 1 (01/01/Number 1/2000 2000): 81. http://ezproxy.gardner-webb.edu/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=edo&AN=ejs998204&site=eds-live.

forecasting three major stocks; their results indicated that the next day's closing price could be predicted within 2% of the actual value of the stock.[19] Post Fama and Jensen literature indicates that the Efficient Market Hypothesis fails to hold when machine learning methods are used and stock markets do not follow random walks.

The second type of financial analyst encourages fundamental macroeconomic analysis of financial environments and has the goal of finding correlations between exogenous variables. This thesis does not address the macroeconomic methods that are used to forecast financial markets. This type of analysis is synonymous with forecasting by humans, but not surprisingly, humans are bias in their predictions. O'Connor, Remus, and Griggs had humans forecast artificial time series to discover if the trend's direction effected forecasting accuracy. They determined that "People were found to be surprisingly inaccurate," and that they had significant difficulties in dealing with downward-sloping series; the study showed that the direction of a time series' trend makes a significant difference in the accuracy of a forecast.[20] Mosteller et al. and Edmendson found that people are able to draw best fit lines which are significantly close to a least-squares regression method, which says that people can assess trends very well.[21] In contrast, Collyer, Standley, and Bowater found that people tended to merely bisect the scatter plots.[22] People are less accurate and efficient when compared to computer centered approaches.

[19] Hassan, Md Rafiul, Baikunth Nath, and Michael Kirley. "A Fusion Model of Hmm, Ann and Ga for Stock Market Forecasting." Expert Systems With Applications 33, no. 1 (2007): 171-80. http://dx.doi.org/10.1016/j.eswa.2006.04.007.

[20] O'Connor, Marcus, William Remus, and Ken Griggs. "Going up-Going Down: How Good Are People at Forecasting Trends and Changes in Trends?" Journal of Forecasting 16, no. 3 (1997): 165. http://ezproxy.gardner-webb.edu/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=bth&AN=9708223194&site=eds-live.

[21] Hoaglin, David C., Frederick Mosteller, and John W. Tukey. "Fitting Straight Lines by Eye." Exploring Data Tables, Trends & Shapes (01// 2006): 225. http://ezproxy.gardner-webb.edu/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=edb&AN=74780855&site=eds-live.

[22] Collyer, Charles E., Kerrie A. Stanley, and Caroline Bowater. "Psychology of the Scientist: Lxiii. Perceiving Scattergrams: Is Visual Line Fitting Related to Estimation of the Correlation Coefficient?" Perceptual and Motor Skills 71, no. 2 (1990): 371-78. http://dx.doi.org/10.2466/PMS.71.5.371-378.

The tertiary view focused on applying time series analysis and machine learning methods based upon rigorously proven mathematics with the intention of forecasting financial returns. Time series analysis methods like ARIMA and GARCH along with machine learning concepts like support vector machines, artificial neural networks, genetic algorithms, fuzzy logic, and more all fall into this category. The results from numerous studies stemming from this school of thought have evidence against the Efficient Market Hypothesis.

Concerning which method of analysis results in higher success, articles published across many journals assert that machine learning models consistently outperform time series analysis models. Hamzaçebi, Akay, and Kutav compared direct and iterative methods classifying with neural networks against multiple methods. They asserted that ARIMA was the worst performing of the seven methods they compared while the artificial neural network performed the best.[23] Yümlü, Gürgen, and Okay used four different models including EGARCH and multi-layered neural network models to predict the ISE-XU-100 daily values, and over the four-years of testing data, EGARCH performed the worst.[24] Hassan, Nath, and Kirley compared a hybrid neural network model against ARIMA when forecasting Apple, IBM, and Dell's stocks; they only tested over five weeks of data, but the neural network model beat ARIMA.[25] Hamzaçebi and Bayramoğlu used ARIMA and artificial neural networks to forecast daily closing prices of the ISE-XU-100, and the neural network produced significantly better results. Even when neural network and time series analysis methods were combined into hybrid models like in Roh's paper, the pure neural network model outperformed the hybrid models.

[23] Hamzaçebi, Coşkun, Diyar Akay, and Fevzi Kutay. "Comparison of Direct and Iterative Artificial Neural Network Forecast Approaches in Multi-Periodic Time Series Forecasting." Expert Systems With Applications 36, no. Part 2 (1/1/2009 2009): 3839-44. http://dx.doi.org/10.1016/j.eswa.2008.02.042.
[24] Yümlü, Serdar, Fikret S. Gürgen, and Nesrin Okay. "A Comparison of Global, Recurrent and Smoothed-Piecewise Neural Models for Istanbul Stock Exchange (Ise) Prediction." Pattern Recognition Letters 26 (1/1/2005 2005): 2093-103. http://dx.doi.org/10.1016/j.patrec.2005.03.026.
[25] Hassan, Md Rafiul, Baikunth Nath, and Michael Kirley. "A Fusion Model of Hmm, Ann and Ga for Stock Market Forecasting." Expert Systems With Applications 33, no. 1 (2007): 171-80. http://dx.doi.org/10.1016/j.eswa.2006.04.007.

[26] The list of articles which detail neural networks forecasting more accurately than ARIMA, GARCH, or similar methods are too numerous to list in this thesis. Since academic literature has empirically determined the success of utilizing artificial neural networks, the focus changes to determining the optimal neural network model for forecasting daily closing prices.

Most of the benefits of neural networks come from rigorously proven theorems about the abilities and limitations of neural networks of different sizes. The most important theorem is the Universal Approximation Theorem which says that any continuous function can be uniformly approximated by a continuous neural network having only one internal hidden layer and with an arbitrary continuous sigmoidal nonlinearity.[27] Because of this theorem and from the successes of other researchers, a single hidden layer neural network is used for financial forecasting of short term stock returns.

There are two different approaches one may take when training and testing an artificial neural network. The direct method trains and validates on historical data and does not update the weights of the network during testing, while the iterative method differs by updating the network to reflect new information after the day is forecasted. A Bayesian would say that any new information would only benefit the accuracy of the network and should be utilized, but what other researchers found contradicts this belief. Hamzacebi, Akay, and Kutay compared direct and iterative methods by creating neural networks for both approaches and forecasting stock returns; they determined that "the direct method is superior." [28] Zhang supports these

[26] Hyup Roh, Tae. "Forecasting the Volatility of Stock Price Index." Expert Systems With Applications 33 (1/1/2007 2007): 916-22. http://dx.doi.org/10.1016/j.eswa.2006.08.001.
[27] Cybenko, G. "Approximation by Superpositions of a Sigmoidal Function." Mathematics of Control, Signals & Systems 2, no. 4 (12// 1989): 303. http://ezproxy.gardner-webb.edu/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=edb&AN=71058362&site=eds-live.
[28] Hamzaçebi, Coşkun, Diyar Akay, and Fevzi Kutay. "Comparison of Direct and Iterative Artificial Neural Network Forecast Approaches in Multi-Periodic Time Series Forecasting." Expert Systems With Applications 36, no. Part 2 (1/1/2009 2009): 3839-44. http://dx.doi.org/10.1016/j.eswa.2008.02.042.

results with his conclusion that the direct method is better for forecasting.[29] In contrast, Weigend, Huberman, and Rumelhart showed that iterative forecasting results in better accuracy when analyzing the sunspot data set.[30] Intuitively, an iterative method would be necessary for real time forecasting, but it may not matter for daily forecasting. For simplicity, the direct method is used for forecasting daily stock returns and is explained thoroughly in chapter seven.

Artificial neural networks are trained with the specific goal of level estimation or classification, where a estimation network forecasts the magnitude of a stock, and a classifier network predicts the sign of change. Some investors only care about the direction of change, others will want to know the magnitude of change, and the rest will want to know both. The question naturally arises, which method is more accurate and does any success translate to profit? Leung, Daouk, and Chen compared multiple models and developed threshold trading rules with the goal of profiting. Their experiment suggested that the classification models outperform the level estimation models in terms of predicting the direction of the stock market movement and maximizing returns from investment trading.[31] Kara, Boyacioglu, and Baykan also developed two efficient models for classification and level estimation, and they concluded that the classification model outperformed the level estimation model in terms of predicting the direction of the stock market movement and maximizing returns from investment trading.[32]

The classification neural network method can be thought of as a specialized level

[29] Zhang, Xiru. "Time Series Analysis and Prediction by Neural Networks." Optimization Methods and Software 4, no. 2 (01/01/Number 2/January 1994 1994): 151. http://ezproxy.gardner-webb.edu/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=edo&AN=ejs11809694&site=eds-live.

[30]Weigend, A S, B A Huberman, and D E Rumelhart, "Predicting Sunspots and Exchange Rates with Connectionist Networks." Non-linear Modelling and Forecasting, SFI Studies in the Sciences of Complexity, Proceedings, 1992.

[31] Leung, Mark T., Hazem Daouk, and An-Sing Chen. "Forecasting Stock Indices: A Comparison of Classification and Level Estimation Models." International Journal of Forecasting 16 (1/1/2000 2000): 173-90. http://dx.doi.org/10.1016/S0169-2070(99)00048-5.

[32] Kara, Yakup, Melek Acar Boyacioglu, and Ömer Kaan Baykan. "Predicting Direction of Stock Price Index Movement Using Artificial Neural Networks and Support Vector Machines: The Sample of the Istanbul Stock Exchange." Expert Systems With Applications 38 (5/1/May 2011 2011): 5311-19. http://dx.doi.org/10.1016/j.eswa.2010.10.027.

estimation neural network because the level estimation network will forecast the magnitude and indirectly the direction of change, but the classification network cannot do both. Since artificial neural networks are especially good at forecasting nonlinear spikes and crashes, level estimation is used for the models presented in chapter seven. The classification and level estimation proficiency of these models are quantified in chapter eight and discussed in chapter nine.

If profitability is the main goal, many researchers are able to forecast the sign of change for daily closing prices significantly higher than the Efficient Market Hypothesis' implied 50% accuracy. Kara, Boyacioglu, and Baykan created support vector machine and artificial neural network models for forecasting the ISE National 100 Index and achieved classification accuracies of 71.52% and 75.74% on average, respectively. Diler used technical indicators to train his back-propagation artificial neural network, and his results showed that the direction of the ISE National 100 Index could be forecasted with a rate of 60.81%. [33] Rodriguez, Gonzalez-Martel, and Sosvilla-Rivero used neural networks to determine the profitability of trading in security markets. Their results indicated that the neural network based trading strategies they developed, when applied to the General Index of the Madrid stock exchange, are always superior to a buy-and-hold strategy for bear and stable markets; this was in the absence of trading costs. [34]

If used correctly, artificial neural networks are proficient at classification and level estimation for many different domains, but they have disadvantages as well. Guresen and Kayakutlu said that artificial neural networks are popular for complex financial markets, but

[33] Diler, A I., "Predicting direction of ISE national-100 index with back propagation trained neural network." Journal of Istanbul Stock Exchange 7 (2003): 65-81.
[34] Fernández-Rodríguez, Fernando, Christian González-Martel, and Simón Sosvilla-Rivero. "On the Profitability of Technical Trading Rules Based on Artificial Neural Networks:. Evidence from the Madrid Stock Market." Economics Letters 69 (1/1/2000 2000): 89-94. http://dx.doi.org/10.1016/S0165-1765(00)00270-6.

they claim that noise caused by changes in market conditions makes it hard to reflect the market variables directly into the models without any assumptions. [35] Similarly, Kim asserted that "ANN often exhibits inconsistent and unpredictable performance on noisy data," and stock markets are noisy systems. [36] Other issues with neural networks come from their ability to learn trained data extremely well but this does not imply *good* results from testing with unseen data. Ticknor says that one drawback of using standard back-propagation networks is the potential for overfitting the training data set, which results in reduced accuracy on unknown test sets. [37]

Another negative aspect of back-propagation trained neural networks is the amount of parameters which must be optimized for gradient descent to converge to a local or global minimum solution. For back-propagation, this includes the learning rate, hidden-layer nodes, momentum, activation function, weight optimization method, and any additional parameters that optimization method uses. Li, et. al compared extreme learning machines to back-propagation trained neural networks. Back-propagation theory stresses that the parameters of hidden-layer nodes needs to be greater than or equal to the input nodes which is 1,011 features in their problem. They found that after several trials of running, it was impossible for their servers to support that many nodes since back-propagation training required too much memory. [38] The authors proposed an extreme learning machine model for forecasting stock returns instead of the traditional back-propagation neural networks.

The subject of debate within this paper is whether extreme learning machines offer a

[35] Güreşen, Erkam and Gülgün Kayakutlu. "Forecasting Stock Exchange Movements Using Artificial Neural Network Models and Hybrid Models." Intelligent Information Processing IV (01//2008): 129. http://ezproxy.gardner-webb.edu/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=edb&AN=76873933&site=eds-live.
[36] Kim, Kyoung-jae. "Financial Time Series Forecasting Using Support Vector Machines." Neurocomputing 55 (1/1/2003 2003): 307-19. http://dx.doi.org/10.1016/S0925-2312(03)00372-2.
[37] Ticknor, Jonathan L. "A Bayesian Regularized Artificial Neural Network for Stock Market Forecasting." Expert Systems With Applications 40 (10/15/15 October 2013 2013): 5501-06. http://dx.doi.org/10.1016/j.eswa.2013.04.013.
[38] Li, Xiaodong, Haoran Xie, Ran Wang, Yi Cai, Jingjing Cao, Feng Wang, Huaqing Min, and Xiaotie Deng. "Empirical Analysis: Stock Market Prediction Via Extreme Learning Machine." Neural Computing & Applications 27, no. 1 (2016): 67-78. http://dx.doi.org/10.1007/s00521-014-1550-z.

significant advantage over the traditional method of forecasting with back-propagation trained neural networks. The extreme learning machine model was originally proposed by Huang et al. from Nanyang University in Singapore. [39] Literature detailing the application of extreme learning machines is recent to this decade, and researchers attest to their proficient ability to forecast in financial markets when compared with back-propagation neural networks. Milacic, Vujovic, and Miljkovic purposed their research towards comparing extreme learning machines and back-propagation neural networks to forecasting gross domestic product growth rate; the authors determined that extreme learning machines had lower root-mean-squared error, higher correlation between the actual and forecasted series, and that "The extreme learning machine algorithm can be effectively utilized in GDP applications and particularly in the GDP estimations." [40]

Extreme learning machines require significantly fewer computations and train faster than back-propagation neural networks. Li, et al. compared hybrid radial basis function extreme learning machines to back-propagation networks when they were applied towards using news articles and the iterative level estimation method. Their results showed that the "RBF-ELM achieved higher prediction accuracy and faster prediction speed when compared to BP-NN." [41] Their article represents the new direction of machine learning and financial forecasting because news articles are now being exploited for high-frequency trading. Incremental learning, error reduced model selection, and online learning algorithms exist for extreme learning machines have already been rigorously proven and provided by Feng, et al. and Liang,

[39] Huang, Guang-Bin, Qin-Yu Zhu, and Chee-Kheong Siew. "Extreme Learning Machine: Theory and Applications." Neurocomputing 70 (1/1/2006 2006): 489-501. http://dx.doi.org/10.1016/j.neucom.2005.12.126.
[40] Milačić, Ljubiša, Srđan Jović, Tanja Vujović, and Jovica Miljković. "Application of Artificial Neural Network with Extreme Learning Machine for Economic Growth Estimation." Physica A: Statistical Mechanics and its Applications 465 (1/1/1 January 2017 2017): 285-88. http://dx.doi.org/10.1016/j.physa.2016.08.040.
[41] Li, Xiaodong, Haoran Xie, Ran Wang, Yi Cai, Jingjing Cao, Feng Wang, Huaqing Min, and Xiaotie Deng. "Empirical Analysis: Stock Market Prediction Via Extreme Learning Machine." Neural Computing & Applications 27, no. 1 (2016): 67-78. http://dx.doi.org/10.1007/s00521-014-1550-z.

et al. [42,43] The potential for improving the training and testing time of big data problems are only starting to be explored, and more information can be found at the academic home page of Guang-Bin Huang. [44,45]

Different beliefs regarding forecasting, methods of forecasting, and the results from academic studies of financial markets have been presented within this chapter. The Efficient Market and Random Walk Hypotheses are rejected in favor of the tertiary belief that machine learning models can detect nonlinear patterns in stock market time series. The experiment proposed within chapter seven, recorded in chapter eight, and discussed in chapter nine propose that these nonlinear patterns can be useful for forecasting daily closing prices of blue chip stocks. Recent research suggests that extreme learning machines could be significantly more useful for short-term financial forecasting than back-propagation neural networks. The theory of both optimization algorithms are presented in the following two chapters.

5 Error Back-Propagation with Stochastic Gradient Descent

5.1 Introduction

The most popular method for optimizing the weights of a neural network is called error back-propagation. Back-propagation is a proven method and has been applied within many different domains but most notably for deep learning. In March 2016, a neural network called Alpha Go, which was trained with a deep learning method, beat Lee Sedol, a 9-dan professional, at the board game Go without any handicaps. Many researchers thought that beating a master Go player was still decades away, but AlphaGo has proven to be a master

[42] Guorui, Feng, Huang Guang-Bin, Lin Qingping, and Robert Gay. "Error Minimized Extreme Learning Machine with Growth of Hidden Nodes and Incremental Learning." IEEE Transactions on Neural Networks 20, no. 8 (2009): 1352-57. http://dx.doi.org/10.1109/TNN.2009.2024147.
[43] Liang, N. y., G. b. Huang, P. Saratchandran, and N. Sundararajan. "A Fast and Accurate Online Sequential Learning Algorithm for Feedforward Networks." IEEE Transactions on Neural Networks 17, no. 6 (2006): 1411-23. http://dx.doi.org/10.1109/TNN.2006.880583.
[44] http://www.ntu.edu.sg/home/egbhuang/reference.html
[45] http://www.ntu.edu.sg/home/egbhuang/index.html

against many internationally ranked players. The benchmark reinforced neural networks as a powerful machine learning tool and error back-propagation is a typical method for training neural network models.

Back-propagation itself does not specify the numerical method for updating the weight, for gradient descent is the most common optimization method. Back-propagation uses the gradient descent theory as a method of modifying the weights of a neural network in two stages. For this thesis, they will be called the "Hidden layer stage" and the "Delta rule." Both stages are used to iteratively modify the weights of a neural network in small increments to reduce the training error on a dataset.

5.2 Gradient Descent

Gradient descent is known as a steepest descent method for finding the minimum of a function, which is used for finding the optimal weights for domain specific regression. The gradient is a generalization of the derivative, and if $f(w_1, \ldots, w_n)$ is a differentiable, real-valued function of several variables, then its gradient is the vector whose components are the n partial derivatives of f. In chapter three, a requirement for the activation function was differentiability and this was because of gradient descent's preconditions.

More generally, if the multi-variable function $F(x)$ is defined and differentiable in a neighborhood of a point W, then $F(W)$ decreases fastest if one goes from W in the direction of the negative gradient of F at W, denoted $-\nabla F(W)$. For small enough η, then it follows that $W_{n+1} = W_n - \eta \nabla F(W_n)$ implies $F(W_{n+1}) \leq F(W_n)$ for $n \geq 0$. Within the machine learning community, η is called the learning rate because the constant directly effects the speed at which the network updates its weights.

Basically, the gradient is subtracted to optimally minimize the function and ultimately, minimize the weights W. If one considers a sequence of values $W_0, W_1, \ldots, W_n$ such that

$W_{n+1} = W_n - \eta_n \nabla F(W_n), n \geq 0,$ then $F(W_0) \geq F(W_1) \geq F(W_2) \ldots,$ and hopefully the sequence will converge to a local minimum.. In special cases where F is a convex function, all local minima are global minima, so the gradient converge to the global minimum, which is best weight optimization possible.

The letter W was used in the previous paragraph to denote the weights of a neural network, which is optimized by using a twostep process involving a cost function and the equation of the previous paragraph. The loss function is constructed for measuring the error of a neural network's forecasts on the training set. Given forecasts $\hat{y}$, real values y, and weights W, then the loss function is expressed as

$L(\hat{y}, y, W) = \frac{1}{2} ||\hat{y} - y||_2^2.$

The benefit of using this function is that the $\frac{1}{2}$ term is canceled by the exponent when the gradient is computed. The next step updates the weights by computing the ∇L_{W_i} individually for the weights of the network, which is explained further in the next section.

$$W_{i+1} = W_i - \eta \nabla L_{W_i}$$

5.3 Hidden Layer Stage and the Delta Rule

Back-propagation is a supervised learning method, so a neural network learns the data set iteratively. The first step involves taking a training example and propagating it though the network to compute the actual error term $\delta_j = (g(x) - y)$ of the output nodes. This process is done as explained in chapter 3 where the weights of a neural network are randomly assigned from an interval, and the output of all neurons towards the output nodes are computed to receive the classification. For multiple nodes, the actual error terms would be $\delta_1, \delta_2, \ldots, \delta_j$, and using an superscript signifies which layer the delta is from. For example, if δ_j^3 refers to the output layer, then δ_j^2 would refer to actual error of the last hidden layer at node j.

The back-propagation process starts when a layer's deltas are calculated using the next layer's deltas. If $\odot$ represents the element-wise multiplication (Hadamard product), l refers to the layer, a^l refers to the activation of a later, $*$ represents the dot product, and w_l are the weights, then process of computing these error terms δ_j^l is described as

$$\delta_j^l = \left((w_l)^T * \delta_j^{l+1}\right) \odot \left(a^l \odot (1 - a^l)\right)$$

Finally, this result is used to calculated the gradient of the loss function for each weight within the network.

$$\frac{\partial L}{\partial w_j^l} = \delta^{l+1} * a^l$$

The gradients of these weights are used within the gradient descent algorithm to modify the weights and achieve a local minimum or global minimum. The point of this process is to determine how much error each individual weight contributed, and modify them in the direction negative to the gradient proportionally to the error each weight contributed.

5.4 Optimizing Learning

There are a few parameters to choose like the learning rate η, momentum ω, regularization constant α, and the batch size for stochastic gradient descent. Depending on their values, this will decide the success of the neural network. These parameters must be found empirically or exhaustively by varying the parameters across numerous training sessions and recording the best set of parameters.

From previous experiments, if the learning rate is too fast, then gradient descent diverges, but if the learning rate is too slow, then gradient descent will not converge to an optimal solution; as mentioned in section 5.2, the learning rate η can be adaptive depending on how quickly the loss function decreases. The number of hidden layer nodes can be chosen according to Kolmogorov-Arnold Representation Theorem as mentioned previously in chapter 3.

The two-phase method proposed within section 5.2 and 5.3 is typically further optimized by adding a regularization term to the loss function and a momentum to the weight update function. The regularization term penalizes larger weights, and tries to create a set of weights that fit the training set but possess the smallest Euclidian norm possible. Regularization is used to solve the problem of overfitting a training set because if the neural network is optimized extremely well for the data set, then the neural network will not perform as well on data it has not seen before. In this sense, the goal of regularization is to increase the generalization of the neural network. To achieve this goal, the term $\alpha\|W\|_2^2$ is added to the loss function. The parameter α is found experimentally by creating, testing the neural network, and varying α.

$$L(\hat{y}, y, W) = \frac{1}{2}\|\hat{y} - y\|_2^2 + \alpha\|W\|_2^2$$

The momentum term increases the speed of convergence of the neural network, which can help the gradient descent optimization process find a better solution (lower minimum). To achieve faster convergence, the difference between the previous sets of weights Δw along with a momentum constant ω is used to accelerate the learning process.

$$W_{i+1} = W_i - \eta \nabla L_{W_i} + \omega \Delta W$$

At this point, the entire algorithm for gradient descent has been described, but this is not exactly how the back-propagation neural networks were optimized for this thesis. The only way that "Stochastic gradient descent" differs from regular gradient descent is that training examples are randomly chosen from the training set in batches. A concise software implementation of this optimization process is given within section 5.5.

5.5 Algorithm

Given a training set and set of weights, there are two phases for training a neural network: propagation and weight update.

Phase 1: Propagation

- Forward propagate the training input through the neural network to generate the network's output values.
- Backward propagate all the propagation's output activations through the neural network for generating the deltas of all output and hidden neurons.

Phase 2: Weight Update

- For each weight, the weight's output delta and input activation are multiplied to find the gradient of the weight.
- The weight is updated by subtracting a percentage of the weight's gradient, and a new weight is formed for the next iteration.

6 Extreme Learning Machines

6.1 Introduction

There is always a need for new artificial intelligence models, which have unique features and are promising for classification and regression. In 2004, Guang-Bin Huang, et. al, proposed a new approach to neural networks and coined its name as extreme learning machine. [46] Although they are new, extreme learning machines and hybrid models have outperformed back-propagation in numerous benchmarks. The difference between the two methods of regression, back-propagation and extreme learning machines are not immediately apparent, but the new algorithm has been shown to learn featured data quicker, with better accuracy, and better generalization than comparative methods.

An extreme learning machines is a type of feed-forward neural network that can be used for regression, classification, and multi-network analysis. The structure of the neural

[46] Huang, Guang-Bin, Qin-Yu Zhu, and Chee-Kheong Siew. "Extreme Learning Machine: Theory and Applications." Neurocomputing 70 (1/1/2006 2006): 489-501.

network follows the same structure as explained within chapter three where there are three layers: input layer, hidden layer, and output layer. Similarly, the weights and connections between nodes are the same as in chapter 3. The difference is, extreme learning machines are initialized with random weights within the input layer, and they are never modified. This method creates independence between the input weights and the output weights of the neural network and allows one to solve an equation for the output weights by linear least squares.

The purpose of this chapter is to present the theory of extreme learning machines and provide rigorous theory along with a nonprofessional's interpretation of the information within the chapter. Within section 6.2, the notation of matrices, Moore-Penrose generalized inverse, and minimum norm least square solutions of general linear systems are provided. Section 6.3 provides a detailed description of the Extreme Learning Machine Algorithm. Section 6.4 discusses forecasting with extreme learning machines. All formulas and theorems are derived from Guang-Bin Huang, et. al. Next, section 6.2.1 will introduce the notation used throughout this chapter.

6.2 Essential Concepts

6.2.1 Linear Algebra Notation

This section is intended for people who know very little about the mathematical notation of linear algebra, and this notation consists of capital letters, the transpose operator, subscripts, and hat notation.

- A upper case letter represents a nxn square matrix of the form:

$$A = \begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,n} \end{bmatrix}$$

- A lower case letter represents a column vector matrix of the form:

$$w = \begin{bmatrix} w_{1,1} \\ \vdots \\ w_{n,1} \end{bmatrix}$$

- The transpose operator swaps row and column entries of a matrix and is denoted by a superscripted T:

$$w^T = [w_{1,1} \quad \cdots \quad w_{1,n}]$$

- Matrices with subscripts are used to imply a necessity to iterate computations. Subscript notation is used with the intention of avoiding row and column notation from linear algebra. The matrices are denoted by

$$w_i = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}$$

- Hat notation is used to distinguish between the theoretical value and an approximated value of the same variable. For example, β is the theoretical (exact) value while $\hat{\beta}$ is an estimation of the theoretical value.

Lastly, x_i denotes the ELM inputs, t_i denotes the target value, H denotes the hidden layer matrix, w_i denotes the weights between the input and hidden layers, and β_i denotes the weights between the hidden and output layers.

6.2.2 Moore-Penrose Generalized Inverse

The Extreme Learning Machine problem is posed as solving a single linear algebra equation $H\beta = T$, where H represents the hidden layer matrix, and T is the target values for forecasting. The goal is to find a matrix generalized inverse $H^\dagger$ for the least squares solution $\hat{\beta} = TH^\dagger$.

Consider the linear system

$$Ax = y$$

where A is a $m \times n$ matrix and y is in the range space of A, written $y \in R(A)$. If A is a nonsingular matrix, then $x = yA^{-1}$ is the exact solution to $Ax = y$. If A is singular, then the Moore-Penrose pseudoinverse is used to compute a least squares solution of the linear system; this is the likely case within machine learning.

Definition 6.1: A matrix G of order $n \times m$ is the Moore-Penrose generalized inverse of a matrix A of order $m \times n$, if the following conditions are met

$$AGA = A$$

$$GAG = G$$

$$(AG)^T = AG$$

$$(GA)^T = GA$$

The same methodology can be applied to the system $H\beta = T$ to receive the least squares solution $\hat{\beta} = TH^{\dagger}$. The variables $A, x,$ and y are used in section 6.2.2, but the reader should note that $H, \beta,$ and T can be used without loss of generality. For simplicity, let $A^{\dagger}$ denote the pseudoinverse in section 6.2.3.

6.2.3 Minimum Norm Least Square Solution of General Linear Systems

Modeling real life phenomena with linear algebra is useful, but an exact solution to the linear system may not exist. Also, the dimensionality of the linear system might be so large that estimating is the only option, which requires solving a linear system for the least-squares solution.

For a general linear system $Ax = y$, we say that $\hat{x}$ is a least-squares solution if

$$||A\hat{x} - y|| = \min||Ax - y||$$

for all x, where $||\cdot||$ is the Euclidean norm.

Definition 6.2: $x_0 \in \mathbb{R}^n$ is said to be a minimum norm least-squares solution of a linear system $Ax = y$ if for any $y \in \mathbb{R}^m$

$$\|x_o\| \leq \|x\|, \forall x \in \{x: \|Ax - y\| \leq \|Az - y\|, \forall z \in \mathbb{R}^n\}$$

The above definition is a mathematical way of saying x_0 is the minimum norm least-squares solution of a linear system $Ax = y$ if x_0 has the smallest Euclidian norm among all the least-squares solutions.

The significance of this definition is not intuitive, but in simply put, having a minimum norm least-squares solution $\hat{\beta}$ will improve the generalization of a neural network. Recall, this was like the regularization term $\alpha\|w\|_2^2$ included within the stochastic gradient descent algorithm. The purpose of the regularization term is to penalize the size of the weights, which is very similar to solving for the minimum norm least-squares solution $\hat{\beta}$.

Theorem 6.1: Let there exist a matrix G such that Gy is a minimum norm least-squares solution of a linear system $Ax = y$. Then it is necessary and sufficient that $G = A^{\dagger}$, the Moore-Penrose generalized inverse of a matrix A.[47]

The minimum norm solution and generalized inverse along with the notation presented previously, are used to develop the theory and algorithms for creating an extreme learning machine in section 6.3.

6.3 Extreme Learning Machine: Approximation Problem

Extreme learning machines take advantage of the independent weight matrices separated by the hidden layer so that a minimum least-squares solution $\hat{\beta}$ is found between the hidden and output layers. The solution is deterministic, results in a fast training algorithm, and the number of hidden layer nodes $\widetilde{N} \leq N$, the number of training samples.

[47] Serre, D. "Matrices Theory and Applications." Graduate Texts in Mathematics - New York, no. 216 (// 2002): ALL. http://ezproxy.gardner-webb.edu/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=edsbl&AN=RN120721940&site=eds-live.

For N arbitrary distinct samples $(x_i,\ t_i)$ where $x_i = [x_{i1}, x_{i2}, \dots, x_{in}] \in \mathbb{R}^n$ and $t_i = [t_{i1}, t_{i2}, \dots, t_{im}] \in \mathbb{R}^m$, extreme learning machines with $\widetilde{N}$ hidden neurons and activation function $g(x)$ are mathematically modeled as

$$\sum_{i=1}^{\widetilde{N}} \beta_i g\left(w_i \cdot x_j + b_i\right) = o_j \ \ for\, j = 1, \dots, N.$$

where $w_i = [w_{i1}, w_{i2}, \dots, w_{in}]^T$ is the weight vector connecting the i^{th} hidden neuron and the input neurons, $\beta_i = [\beta_{i1}, \beta_{i2}, \dots, \beta_{im}]^T$ is the weight vector connecting the i^{th} hidden neuron and the output neurons, b_i is the threshold of the i^{th} hidden neuron, and $w_i \cdot x_j$ denotes the inner product of w_i and x_j. As mentioned earlier in chapter 3, the activation function for the output layer is the linear function $f(x) = x$. The bias has not been previously introduced, but in a two-dimensional space, the bias is the option to offset the y-intercept of a decision boundary.

A single hidden-layer feed-forward neural network with $\widetilde{N}$ hidden neurons and activation function $g(x)$ can approximate these N samples with zero error, implying $\sum_{i=j}^{N} \left\| o_j - t_j \right\| = 0$. There exist β_i, w_i, and b_i such that

$$\sum_{i=1}^{\widetilde{N}} \beta_i g\left(w_i \cdot x_j + b_i\right) = t_j \ \ for\ \ j = 1, \dots, N.$$

The above N equations are written compactly as

$$H\beta = T$$

where H is called the hidden layer output matrix of the neural network. The i^{th} column of H is the i^{th} hidden neuron's output vector with respect to the inputs $x_1, x_2, \dots, x_N$.

$$H(w_{\widetilde{N}}, b_{\widetilde{N}}, x_N) = \begin{bmatrix} g(w_1 \cdot x_1 + b_1) & \cdots & g(w_{\widetilde{N}} \cdot x_1 + b_{\widetilde{N}}) \\ \vdots & \ddots & \vdots \\ g(w_1 \cdot x_N + b_1) & \cdots & g(w_{\widetilde{N}} \cdot x_N + b_{\widetilde{N}}) \end{bmatrix}_{N \times \widetilde{N}}$$

$$\beta = \begin{bmatrix} \beta_1^T \\ \vdots \\ \beta_{\tilde{N}}^T \end{bmatrix}_{\tilde{N}\times m}$$

$$T = \begin{bmatrix} t_1^T \\ \vdots \\ t_{\tilde{N}}^T \end{bmatrix}_{N\times m}$$

From the previously stated theorem 6.1, the minimum norm least-squares solution of the above linear system is

$$\hat{\beta} = H^{\dagger}T$$

As a result, Huang, et al. claim that the following properties hold:

1. Minimum training error. The special solution $\hat{\beta}$ satisfies the equation

$$\left\|H\hat{\beta} - T\right\| = \left\|HH^{\dagger}T - T\right\| = \min_{\beta} \|H\beta - T\|$$

2. Smallest norm of weights and best generalization performance.
3. The minimum norm least-squares solution of $H\beta = T$ is unique, which is $\hat{\beta} = H^{\dagger}T$.

6.4 Extreme Learning Machine Algorithm

The algorithm for training the extreme learning machine for forecasting is given below.

Given a training set $\aleph = \{(x_i, t_i) | x_i \in \mathbb{R}^n,\ t_i \in \mathbb{R}^m, i = 1, \ldots, N\}$, activation function $g(x)$, and hidden neuron number $\tilde{N}$,

- Assign arbitrary input weight w_i and bias b_i, $i = 1, \ldots, \tilde{N}$
- Calculate the hidden layer output matrix H.
- Calculate the output weight β:

$$\beta = H^{\dagger}T$$

where H, β, and T are defined as above in section 6.3.

Simply speaking, β provides the domain specific regressions by being the best solution between the hidden-matrix H, which encodes the features of a data set, and the target values. Within stock market forecasting, the H matrix would encode the features, ratios, or any relevant correlations between the high, open, low, close, volume, or any other inputs into the model. The target matrix is the closing values of the stock offset forward one day, and the β weights would provide the best method to transform H into T. The implementation of stock market forecasting with extreme learning machines and stochastic gradient descent is described in chapter 7.

7 Methods

7.1 Introduction

While the theory of artificial neural networks is based upon mathematics, the application is implemented with programming languages. With computer's rising computation power and falling cost, the average person has access to hardware that can be applied towards big data problems. When considering market analysis, the stock market is a big data problem since there are decades of daily opening, high, low, closing, adjusted closing, and volume values are taught to artificial neural networks. The market information is often used to create any number of technical indicators. Without algorithmic efficiency, a program using this data may never finish because of the extensive computation and underlying architecture of the neural network. Thankfully, there are efficient implementations of the extreme learning machine models and back-propagation algorithms within multiple different languages.

Both packages are used to obtain results within this thesis are free, open source, and written for the Python programming language. The first package is called *hpelm* which stands for high-performance extreme learning machine, and the package is modeled after the description of extreme learning machines from Guang-Bin Huang of Nanyang Technical

University. [48,49] Within *hpelm* there is support for graphics card accelerated training, iterative validation, multiple transfer functions, batch processing, and file operations which are described within the author's academic paper. [50] Overall, the package provides a complete tool set for neural network regression with financial time series. The second package is called Multi-layer-Perceptron Regressor, and the package is a subset of machine learning packages from *scikit-learn.* There is support for multiple different optimization methods including Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm, adam, and stochastic gradient decent. *Scikit-learn* provides the necessary code for predicting financial time series. [51]

7.2 Combining Time Series Analysis and Artificial Neural Networks

When using artificial neural networks for regression, the organization of the data impacts how features are learned. This presents a common optimization question; is there a sufficient or necessary condition for artificial neural networks to find an optimal solution? The answer is that there is no condition which will guarantee that an artificial neural network will perform well on a training or testing set of data, and regression using artificial neural networks is empirical. Luckily, there are twenty years of studies which demonstrate successful methods of financial forecasting; similar theories and methods were adapted for this thesis and are explained in detail below.

For a neural network to be consistently successful at forecasting, the methods of training and testing need to be identical. Within the real world, only previous historical stock data is available to analysts, so when training a neural network, a similar methodology should be followed. Inputs to the artificial neural network are modeled as sequential previous days,

[48] https://pypi.python.org/pypi/hpelm
[49] http://www.ntu.edu.sg/home/egbhuang/
[50] Akusok, A., K. M. Björk, Y. Miche, and A. Lendasse. "High-Performance Extreme Learning Machines: A Complete Toolbox for Big Data Applications." IEEE Access 3 (2015): 1011-25. http://dx.doi.org/10.1109/ACCESS.2015.2450498.
[51] http://scikit-learn.org/stable/index.html

and target values are the future closing prices. For example, if the inputs to the neural network are days 1, 2, 3, 4, and 5, then the corresponding target value the network should learn to predict is the 6th day. The next sections describe how data was structured for forecasting.

7.3 Structuring of Data

Data structuring is an important topic within computer science and with big data collections, correct data structuring is essential. There are many different types of data structures like stacks, lists, queues, linked-lists, dictionaries, and more which are chosen depending on the process to be modeled. The list data structure is a continuous sequence of elements stored within the memory of a computer like shown below.

$$list = [e_1, \quad e_2, \quad \dots, \quad e_M]$$

The list above is the form of an individual training, validation, or testing sample. The training, validation, and testing sets must follow a two dimensional list with the first dimension representing the whole structure and the second dimension representing the specific training, validation, or testing sample. When applied to an artificial neural network with M input nodes, a set of S samples and arbitrary elements e is represented as:

$$input = [[e_1, \dots, e_M]_1, \quad [e_1, \dots, e_M]_2, \quad \dots, \quad [e_1, \dots, e_M]_S]$$

While this list consists of M features and S samples, financial forecasting logically requires the neural network to have an output dimension of 1; this is the next day's closing price. Using a similar naming convention, let the dimensionality of the output layer be 1. Then the target set is represented as:

$$target = [[e_1]_1, \quad [e_1]_2, \quad \dots, \quad [e_1]_S]$$

The historical values are structured this way so that an iterative training method is possible. The first element of the input list corresponds to the first element of the target list, and the training, validating, and testing of neural networks are iteratively computed using these lists. The mathematics term “set” will be used instead of the programming term “list,” but within the context of this paper, they refer to the same objects. Within the next section, the methods of acquiring, scaling the historical data, and formatting are explained.

7.4 Acquiring, Preprocessing, and Formatting Data

A significant part of regression with neural networks involves the creation or acquiring of specially formatted data sets for training, validating, and testing. There are well known sets like the MNIST Handwriting and Digit database, which is applied for classification within images with handwritten words. Within financial forecasting, there are no such extensive databases because of the variability of features which can be used for regression. The training, validation, and testing sets must be acquired, formatted, and scaled by the individual.

Historical stock data can be downloaded from the finance sections of popular websites like *yahoo.com* in the form of *.csv* files, which stands for comma separated values. The *.csv* file’s date range can be specified prior to downloading, and the file contains the date, open, high, low, close, volume, and adjusted close in columns. To perform one testing cycle, *.csv* files for training, validation, and testing need to be downloaded and must be temporally independent. Within this paper, the training set represents all available data from the stock’s start to about four years ago. The next two years are used for the validation set, which is a set for optimizing the parameter selection process. Finally, the test set is taken to be the last two years available. The open, high, low, close, and volume are parsed into their own lists. Before the five lists are transformed into a training set, the individual lists are linearly scaled.

Min-Max Scaling takes a feature’s interval, $[min, max]$, and scales it into a new interval, $[newMin, newMax]$. The sigmoidal and hyperbolic tangent functions have specific ranges, and typically, a feature is scaled so that it lies within $[0, 1]$ or $[-1, 1]$. The minimum and maximum values used for linear scaling come from the training *.csv*. Let x' represent the scaled value and x represent a value of the stock; the equation for Min-Max Scaling is

$$x' = \frac{x - min(x)}{max(x) - min(x)}$$

Scaling the features of the stocks helps to reduce the interference of large inputs when compared to smaller inputs and provides faster convergence of the back propagation algorithm. Empirically, scaling the features results in more accurate predictions because the neural network tends to be less sensitive to sharp changes, reducing the frequency of over forecasting.

Once scaled, the lists are allocated in a special order to represent training, validating, or testing sets. For the neural network to be able to learn the method of prediction, each corresponding value between the input and target lists must be offset. Let “e” be element, “s” be sample, “o” be open, “h” be high, “l” be low, “v” be volume, “c” be close, and let 25 and 1 be the dimensionality of the input and target layer respectively.

Sample	Input List	Target List
1	$[o_1, h_1, l_1, v_1, c_1 \ldots, o_5, h_5, l_5, v_5, c_5]$	$[c_6]$
2	$[o_2, h_2, l_2, v_2, c_2 \ldots, o_6, h_6, l_6, v_6, c_6]$	$[c_7]$
3	$[o_3, h_3, l_3, v_3, c_3 \ldots, o_7, h_7, l_7, v_7, c_7]$	$[c_8]$
. . .	. . .	. . .
m	$[o_{m-4}, h_{m-4}, l_{m-4}, v_{m-4}, \ldots, o_m, h_m, l_m, v_m, c_m]$	$[c_{m+1}]$

Table 1: A software representation of training and testing a model

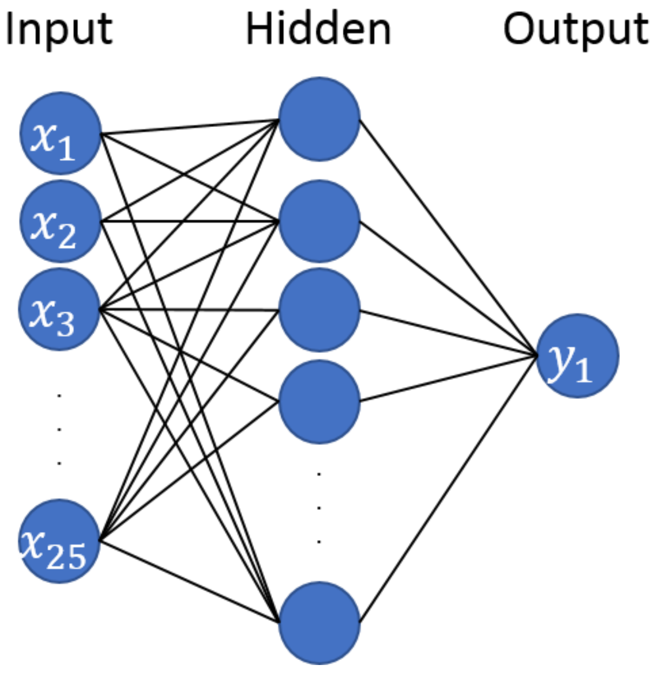


Figure 7.1: A visual representation of table 1, which is the financial forecasting model

7.5 Extreme Learning Machine Algorithms

7.5.1 Model Selection

To forecast stock data, a model must be created, trained, and validated. The first step is to initialize an extreme learning machine object and add hyperbolic tangent nodes too the model. The training set created by the method presented in section 7.4 is added to a queue for training. The model is then trained and validated by first solving the $H\beta = T$ equation, and then the actual error is iteratively computed for many models at different node step sizes; the most accurate amount of nodes is chosen for the testing model. This process takes about two seconds to complete.

7.5.2 Testing

Testing the Extreme Learning Machine model involves loading the testing sets and calling the predict method. The prediction is done in a scaled form, so the known training set max and min along with the scaling interval boundaries are used to finish the forecasting process.

7.6 Stochastic Gradient Descent Algorithms

7.6.1 Model Selection

For consistency, the exact same data sets that were created for the Extreme Learning Machine models are used for the Back Propagation models, and the *.hdf5* files are simply copied to the local directory of the Back Propagation scripts. *Scikit-learn* does not support reading in *.hdf5* files, so they have to be loaded in memory using the *h5py* package. There are four parts of data (training input and target, validation input and target), and they are concatenated together into one input and one target array. After initializing a multi-layer perceptron model, those array are used for training and validating the model.

Because of the numerous adjustable parameters and excessive training time, selecting a model trained with back-propagation is time consuming. Researchers have resorted to experimenting with different learning rates, iterations, momentums, hidden layer sizes, and more to find an acceptable model. The parameters listed within the next section were obtained by running a script hundreds of times and observing the changes while varying parameters. In comparison, a researcher only has to optimize the hidden layer nodes and regularization constant α for an extreme learning machine.

7.6.2 Testing

Testing a neural network trained with back-propagation follows a similar form to testing of an extreme learning machine. The testing input set is used to receive a prediction array y, and it's rescaled using the knowledge of the max, min, and scaling interval from the training set. Following the prediction, the error and graphs can be computed for analysis, which is explained further within chapter eight.

7.6.3 Scripts

The four scripts that were written for stock market forecasting are listed in full at the end of the appendix. The scripts are called "elm_model.py," "mlp_model.py," "train_set_generator.py," and "test_set_generator.py." The first two files create, train, test, and compute error for their respective models. The last two of the files are used to create the data sets which the first two files use.

8 Results

8.1 Design of Training, Validation, and Testing Sets

Table 2: The date rages and samples for the training, validation, and testing sets

Set	Number of Samples	Start Date	End Date
Training	~6000-12000 (varies)	Earliest available day	2/14/2013
Validation	498	2/15/2013	2/14/2015
Testing	498	2/15/2015	2/13/2017

The training, validation, and testing sets have their own role in developing accurate predictions. The neural network learns the training set, and by modifying the weights between different layers teaches the underlying features of the process. Part of the training set is reserved for validation, and the validation set allows for model selection by computing the actual error post-training and exhaustively searching for the optimal set of parameters. Extreme learning machines use the validation set for optimizing the number of nodes in the hidden layer, while stochastic gradient descent uses the validation set to determine when to stop training the neural network. The testing set assesses the neural network's ability to forecast data the model has never seen before. These three sets are required to be disjoint to receive credible results. For example, a professor would not give his student all the questions and answers to a test prior to the actual test.

Usually, the total amount of available data is divided into 80% for training and 20% for testing, where the training set is further divided into 80% for training and 20% for validation.

In contrast, the training set is composed of all available data before the last four years. The validation and testing sets are composed of the most recent four years of data divided into two year blocks. Data for neural networks is usually scarce or has to be created from scratch. This experiment is designed to take advantage of the publicly available information. The table below summarizes the calendar ranges and sample sizes of the training, validation, and testing sets.

8.2 Error Metrics

The only way to measure the success of a neural network is to compare results to previous experiments. Previous experiments are useful for estimating parameters and realizing the current ceiling of success. Researchers measure important components of neural networks to decide which model is better suited for its domain. The error measurements that were used include the number of nodes, training time, mean square error, and two "hit or miss" ratios. The number of nodes corresponds to nodes within the hidden layer. The training time is the amount of time the script takes between beginning and ending the training phase. Mean squared error is the average of squared deviation from the real value of the stock. For n testing samples, real values y_i, and forecasted values $\widehat{y_i}$, mean-squared error is expressed as

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\widehat{y_i} - y_i)^2$$

While mean squared error is useful for comparisons between different models, MSE does not measure the daily direction of change for a stock, so two other "hit or miss" methods were adopted for the purpose of better defining a successful daily forecast; this is discussed further in section 9.2.

8.3 Tabled Results

The exact parameters and results from training, validating, and testing are listed below. The graphs of the real stock values plotted against the running daily forecasts are within the appendix.

Single layer Perceptron Optimization: Stochastic Gradient Descent	
Nodes	40
Activation Function	$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
Learning rate	$\eta = 0.001$
Momentum	$\omega = 0.9$
L2 Regularization term	$\alpha = 0.001$

Extreme Learning Machine Optimization: Least Squares	
Nodes	Dependent on Validation
Activation Function	$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
L2 Regularization term	$\alpha = 1$

Back-propagation with Stochastic Gradient Descent Results: 25 input nodes, 1 output node

Stock	Nodes (tanh x)	Training Time (sec)	Mean Squared Error	Hits/Misses (Type 1)	Hits/Misses (Type 2)
APPL	40	17.7	4.47	283/214	57/82
WMT	40	23.6	0.339	349/148	123/69
BAC	40	61.8	0.06	312/185	91/78
F	40	35.3	0.036	324/173	71/93
KO	40	34.2	0.07	327/170	82/83
Average	**40**	**35.5**	**.995**	**64.2%**	**50.4%**

Extreme Learning Machine Results: 25 input nodes, 1 output node

Stock	Nodes (tanh x)	Training Time (sec)	Mean Squared Error	Hits/Misses (Type 1)	Hit/Misses (Type 2)
APPL	497	1.44	1.454	372/125	124/69
WMT	890	1.63	0.267	370/127	146/51
BAC	440	1.5	0.039	356/141	116/77
F	276	1.38	0.027	366/131	128/58
KO	590	1.91	0.063	363/134	134/45
Average	**538.6**	**1.57**	**0.37**	**73.5%**	**68.4%**

9 Discussion

9.1 Nodes, Training Time, Mean-Squared Error

Model Type	Nodes ($\tanh x$)	Training Time (sec)	Mean-Squared Error (MSE)	Hits/Misses (Type 1)	Hits/Misses (Type 2)
BP-NN	40	35.5	.995	64.2%	50.4%
ELM	**538.6**	**1.57**	**0.37**	**73.5%**	**68.4%**

As seen from the table, extreme learning machines used significantly more nodes than the back-propagation with an average of 13 times more nodes used. The number of nodes for the back-propagation model was chosen based upon the general rule of using $2n + 1$ hidden-layer nodes for n input nodes. Following this guideline, the number of nodes should have been 51, but better generalization was observed at 40 nodes. When a node number similar to the amount extreme learning machines use (100+) was chosen for stochastic gradient descent, the model performed worse than having fewer nodes even when the other parameters like learning rate, momentum, and more were adjusted to account for the increase. In addition to worse accuracy, the training time increased rapidly when the number of nodes increased in the hidden layer.

Because of the few number of parameters and fast training, extreme learning machine models can iteratively compute the error on a validation set for a given step size; the model which has the lowest validation error is chosen for testing. Although the extreme learning machines used more nodes, their training time and mean-squared error were significantly lower when compared to the back-propagation model. Mean-squared error is a common measurement among machine learning models, but it is not very informative for stock market forecasting. For forecasting daily closing prices, many people only care about knowing the sign of change for the next day.

9.2 Directional Change

Since mean-squared error does not accurately represent the interactions between the stock's features and the accuracy of forecasting the next day's closing price, two different error metric were proposed to measure the sign of change. For the first type, a hit signifies that the stock and prediction have the same sign of change between any two adjacent points, while a miss would be the complement of that situation. For the second type, given any three points, if the model is predicting the correct sign of change between points 1 and 2, and the sign of change differs for the real closing price from 1 to 2 and from 2 to 3, then a hit is signified by the model predicting the sign of change between points 2 and 3 correctly. Simply speaking, the second type of hit or miss ratio measures whether a model can accurately forecast a local-min or local-max. The second type of hit or miss ratio was created because the first type might reflect positive results even though the model is projecting data forward (see section below).

The averaged results in the table within section 9.1 show that extreme learning machine could forecast both types of hit or miss ratios for stocks better than the back-propagation model. For the first type, the extreme learning machines forecasted an average of 73.5% and 68.4% while the back-propagation model forecasted an average of 64.2% and 50.4% of the time for types 1 and 2 respectively. As explained within chapter 4, researchers have achieved forecasting accuracies between 60-70%; the Efficient Market Hypothesis implies the maximum accuracy for any daily prediction would be 50%.

9.3 Projecting Data

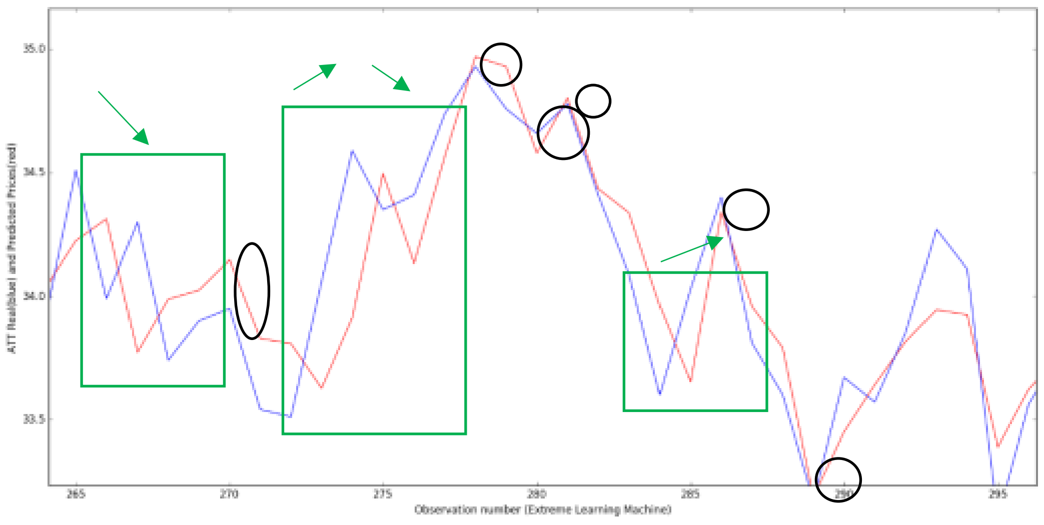


Figure 9.1: Instances of projected time series data.

Both models were observed to project data forward when the previous days' value were forecasted. In this situation, the predicted closing prices (red) look like the forward "projection" of the real closing prices (blue). The instances of these "projections" are outlined with green boxes, and the green arrows point in the direction of projection from the blue line towards the red line. In these situations, the models found a very simple method of producing a close approximation and lowering the training error or finding a least-squares solutions. A model which projects data forward does not learn the underlying nonlinear relationships which lead to accurate classification or level estimation; the model is reacting to the changes of the stock instead of reacting with the changes of the stock. The graph above was an arbitrary choice, and these patterns are seen infrequently within the graphed results.

The projecting of data might not be a problem as one might conclude. Even though both of these models' weights were optimized with different methods, they exhibit projecting previous closing prices forward, which may indicate that the issue is a consequence of the training process or noisy data. Both models correct their forecasting error and recover to accurately forecast inflection points as shown by the black circles. From the previous section, the extreme learning machine model showed that it predicted the sign of change and inflection points with high accuracy, so on average, the model performs very well. The back-propagation model was less successful with forecasting the sign of change and inflection points, and projecting data forward may be the reason why it was not comparatively accurate.

9.4 Threshold Responses

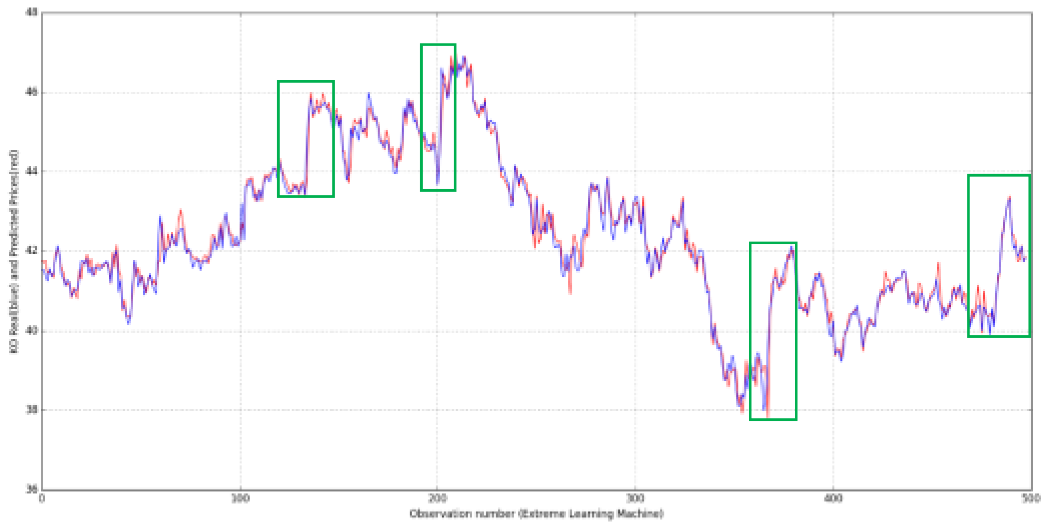


Figure 9.2: Threshold responses in Green boxes:

Artificial neural networks have the advantage of recognizing nonlinear patterns. Humans are not able to recognize these patterns, but neural networks learn nonlinear patterns when training the model with a training set. The type of nonlinear pattern that's easy for anyone to see are threshold responses, which is outlined by the figure above. Consider the value of a stock to be $x_1 = 4.8$, and as that stock starts to increase or decrease incrementally: $x_2 = 4.85, x_3 = 4.9, x_4 = 5.0$. All of the sudden, the value of the stock jumps to $x_5 = 6.0$; this is a threshold response. It's like the "three easy payments of $19.99" methodology used for selling products. To humans, statistically speaking, $19.99 and $20.00 prompt different responses.

More generally, these threshold values create volatile stock markets and are sourced from computer algorithms or humans. By observation, both models accurately forecasted these rapid price changes well within the five testing stocks. The extreme learning machines forecasted these changes better than the back-propagation model. The extreme learning machine model also tended to over predict and under predict these changes less than the back-propagation model. Both models became more proficient at forecasting threshold responses after the training data was preprocessed using min-max linear scaling.

10 Conclusion

The comprehensive mathematical theory of training neural networks was presented within this thesis, but it's necessary to evaluate previous literature along with the results from forecasting. A few questions were posed at the beginning of this thesis including *is a stock market predictable?* and *what's a sufficient method for optimizing a neural network for forecasting these markets?*

While the results of others and this thesis have very positive results on independent data, the academic playground mentality might not transfer well to an actual financial market. There are people who profit utilizing mathematical concepts like artificial neural networks, but they are more than likely using methods in addition to a nonlinear classifier. Also, there's the problem of authors modifying the testing parameters to perform well on the testing set, and therefore the testing parameters were held constant for the neural networks with the hopes of finding a general model that forecasts daily closing prices. In the real world, there is no testing set, and a financial analyst would not be able to compute the actual error of forecasts and choose the appropriate parameters for profiting. It is known that machine learning models scrape news articles, tag parts of speech, and factor that information into their forecasting models; this has been recognized from the speed a stock reacts to news within oil markets, which is fast enough that a human could not have read the article and reacted by the time the stock changed drastically. To conclude, an artificial neural network model represents an important component of a system for financial forecasting daily stock change.

Extreme Learning Machines and Back-propagation with stochastic gradient descent were presented and evaluated against each other. Of the five different stocks both networks were tested on, the extreme learning machine model had better results, and these results are consistent with the research community's findings. The extreme learning machine model had better forecasting speed, less memory usage, and significantly better accuracy. The hit or miss ratio provided within this paper implies that extreme learning machines would perform well with forecasting the direction of change in a market, which is an important factor of successful daily forecasts.

Extreme learning machines are a new method for optimizing neural networks and finding solutions to domain specific problems. While the results of this thesis are positive and interesting, a more important question to ask is whether extreme learning machines are able to outperform deep learning methods like deep reinforcement learning. Hopefully, more research will be published in the future that show case the ability of this recently developed and highly capable method of regression.

11 Appendix

11.1 Graphs (below)

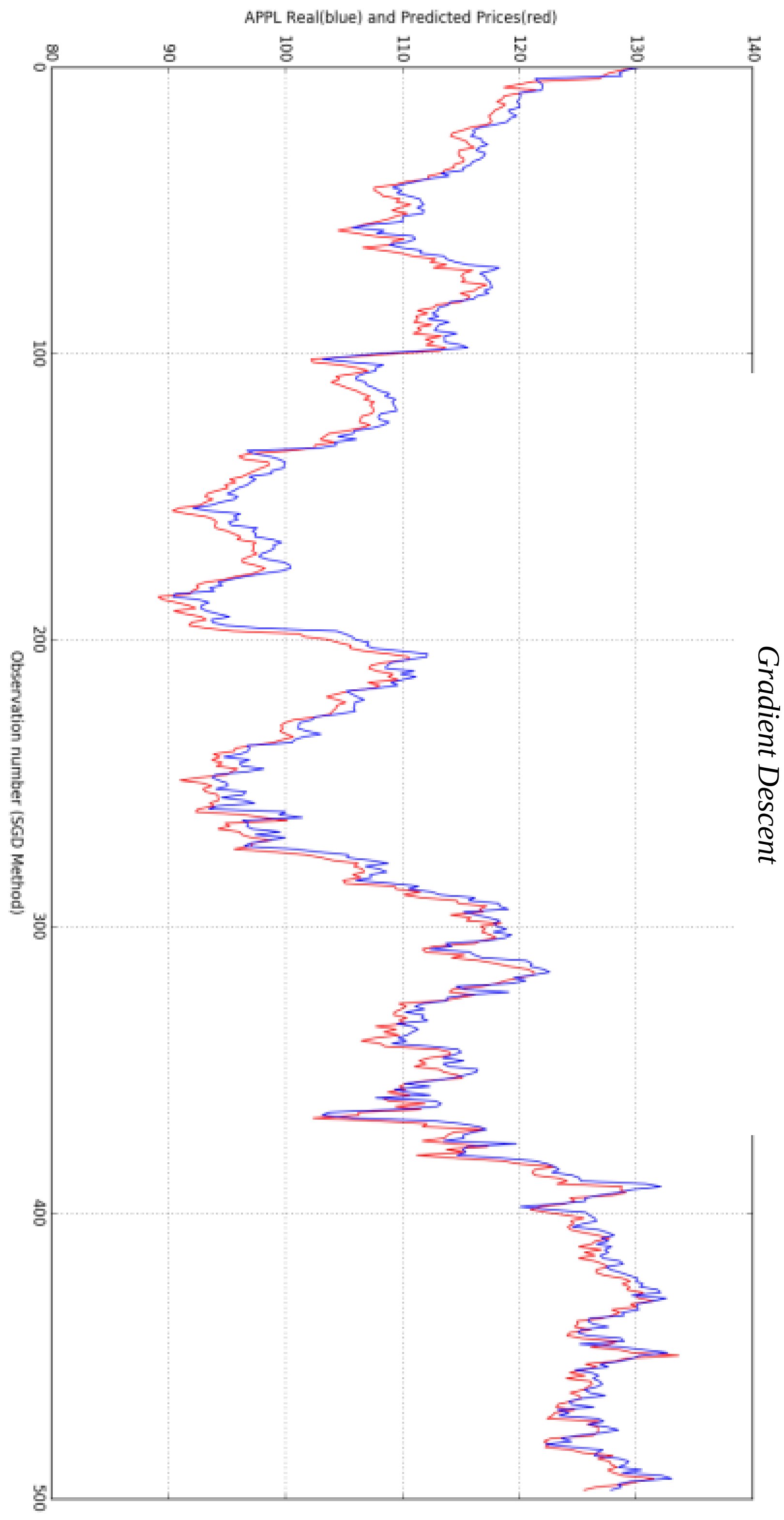


Figure 11.1: Forecasting Apple inc. with Stochastic Gradient Descent

Figure 11.2: Forecasting Bank of America. with Stochastic Gradient Descent

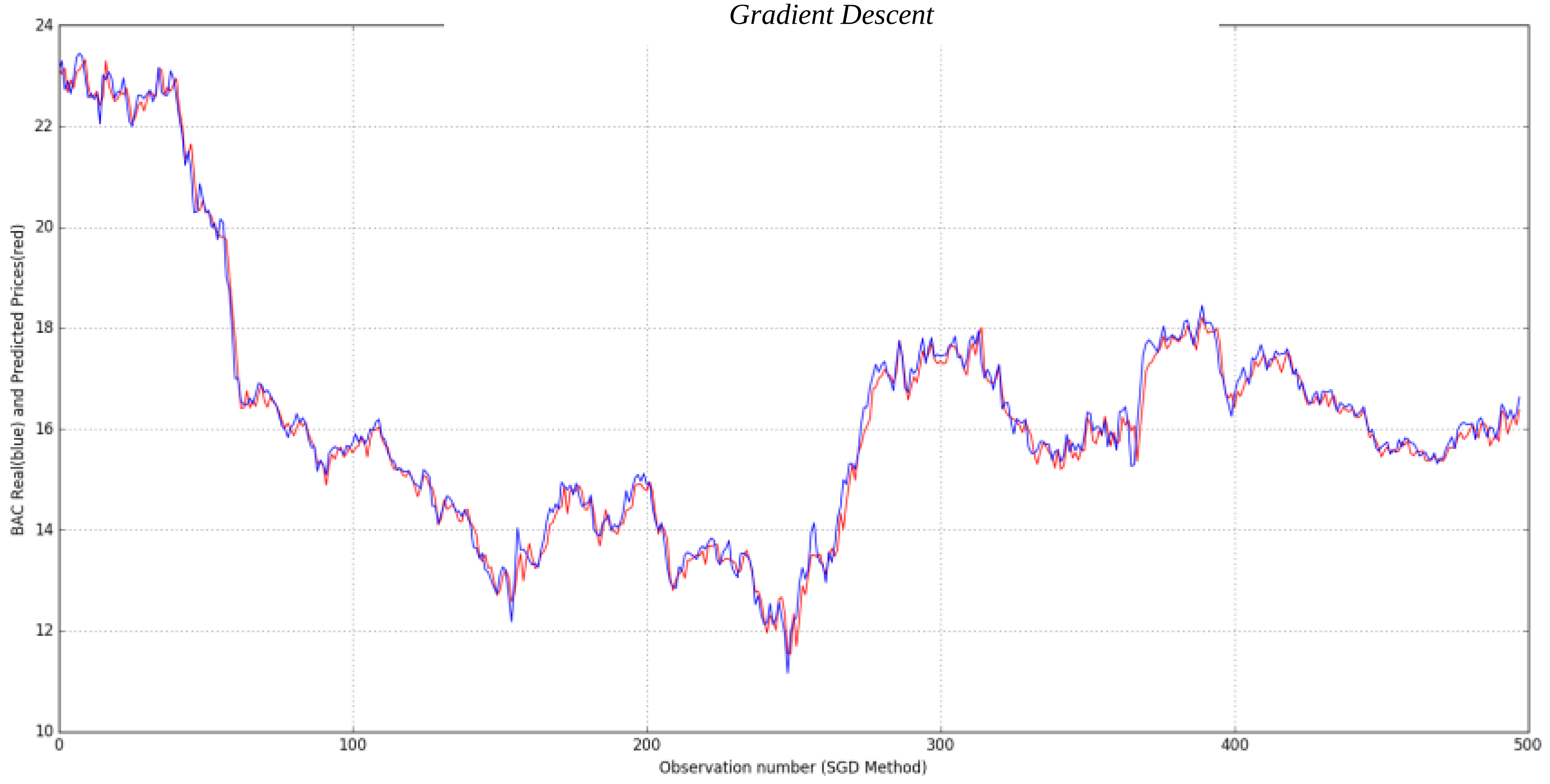


Figure 11.3: Forecasting Ford Motor Company. with Stochastic Gradient Descent

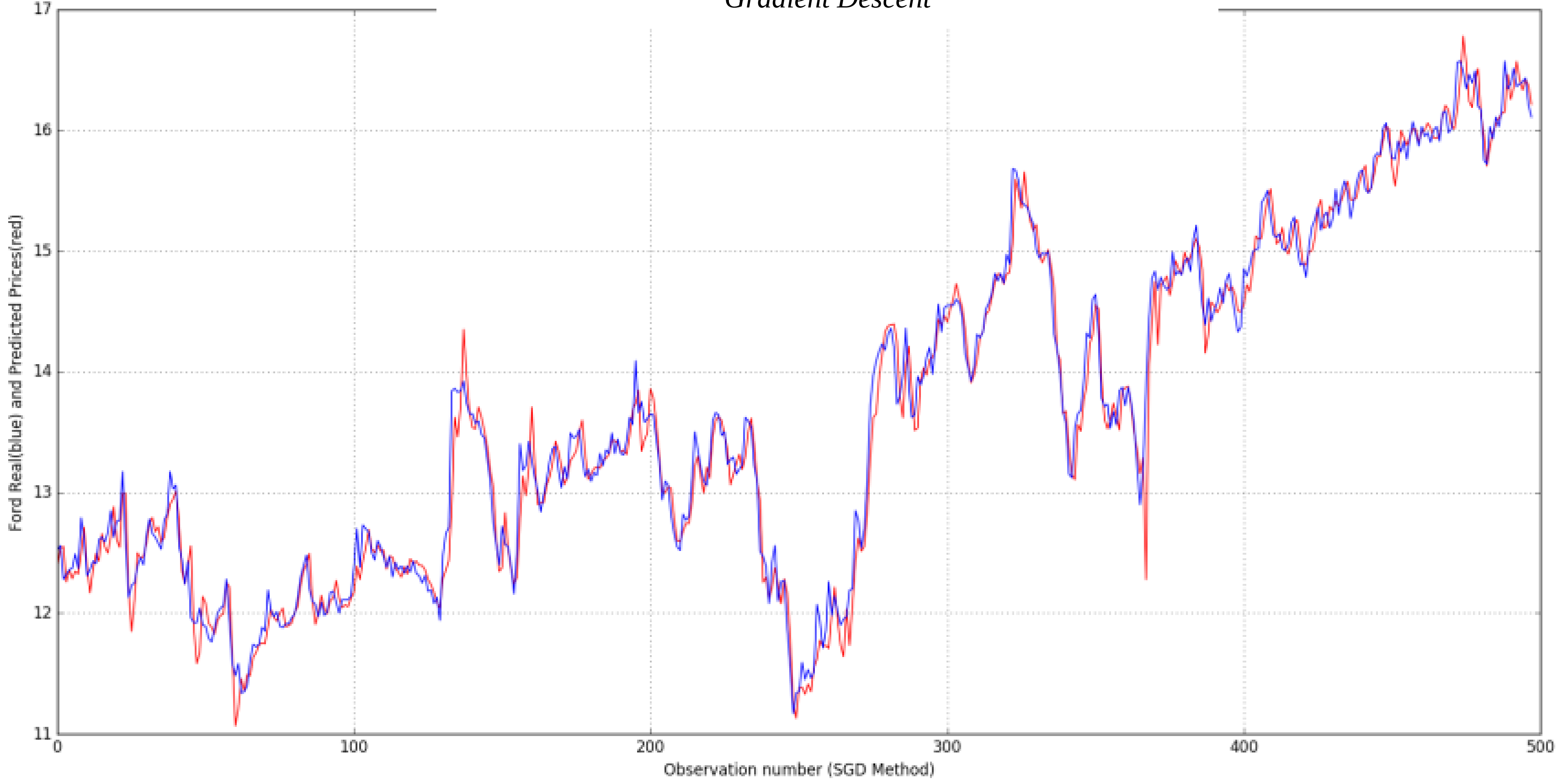


Figure 11.4: Forecasting Coca-Cola with Stochastic Gradient Descent

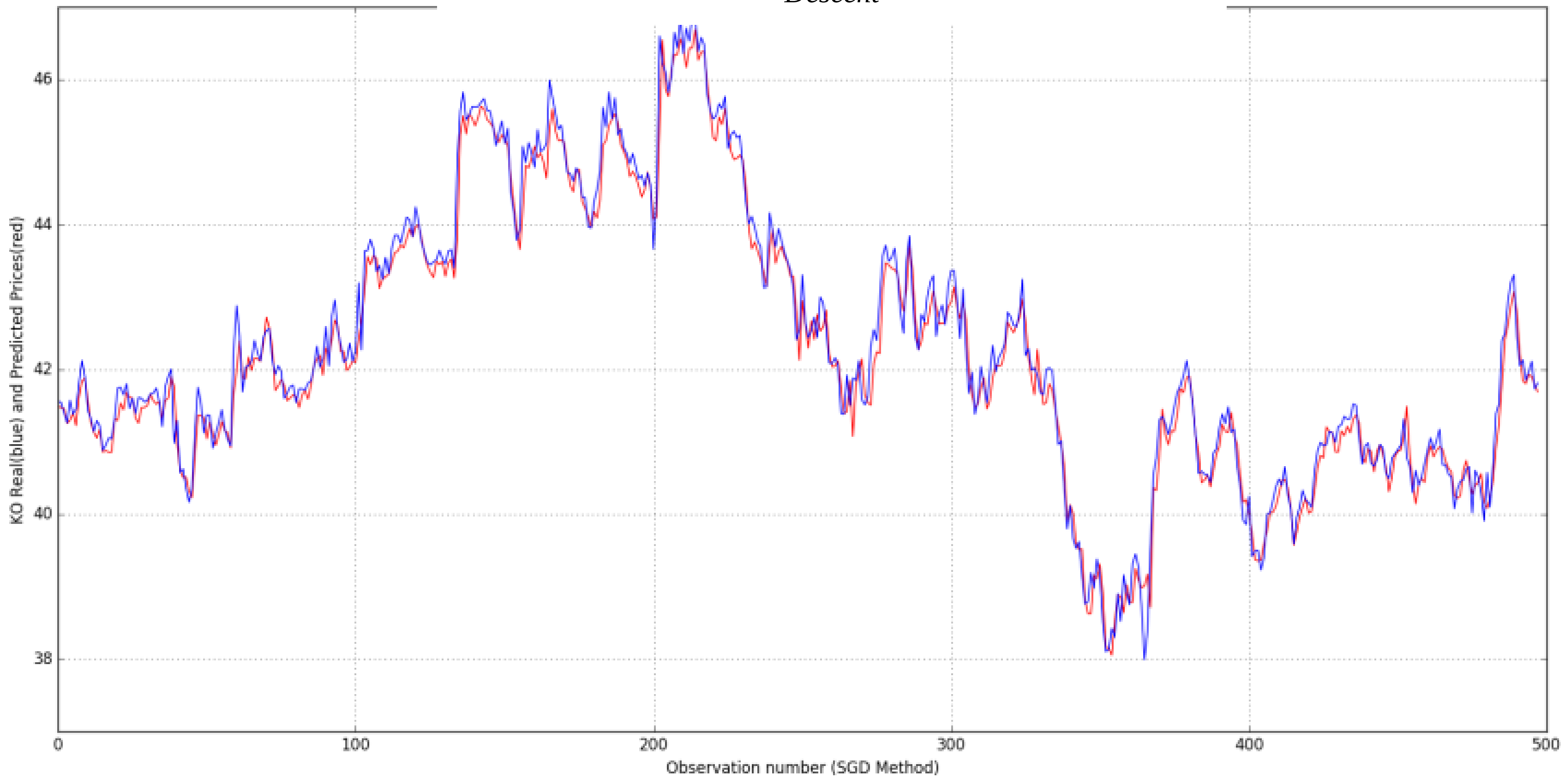


Figure 11.5: Forecasting Walmart with Stochastic Gradient Descent

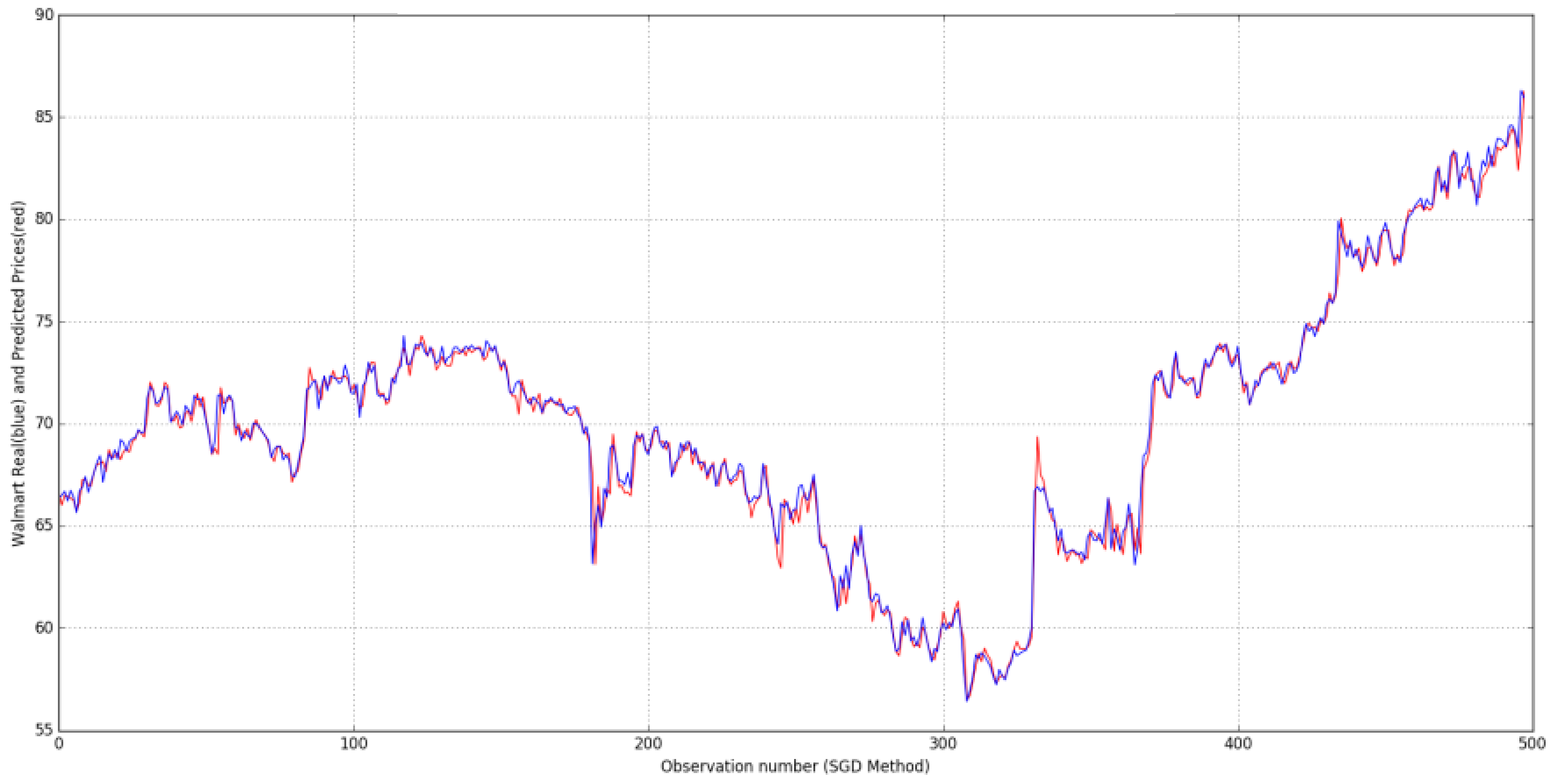


Figure 11.6: Forecasting Apple inc. with Extreme Learning Machines

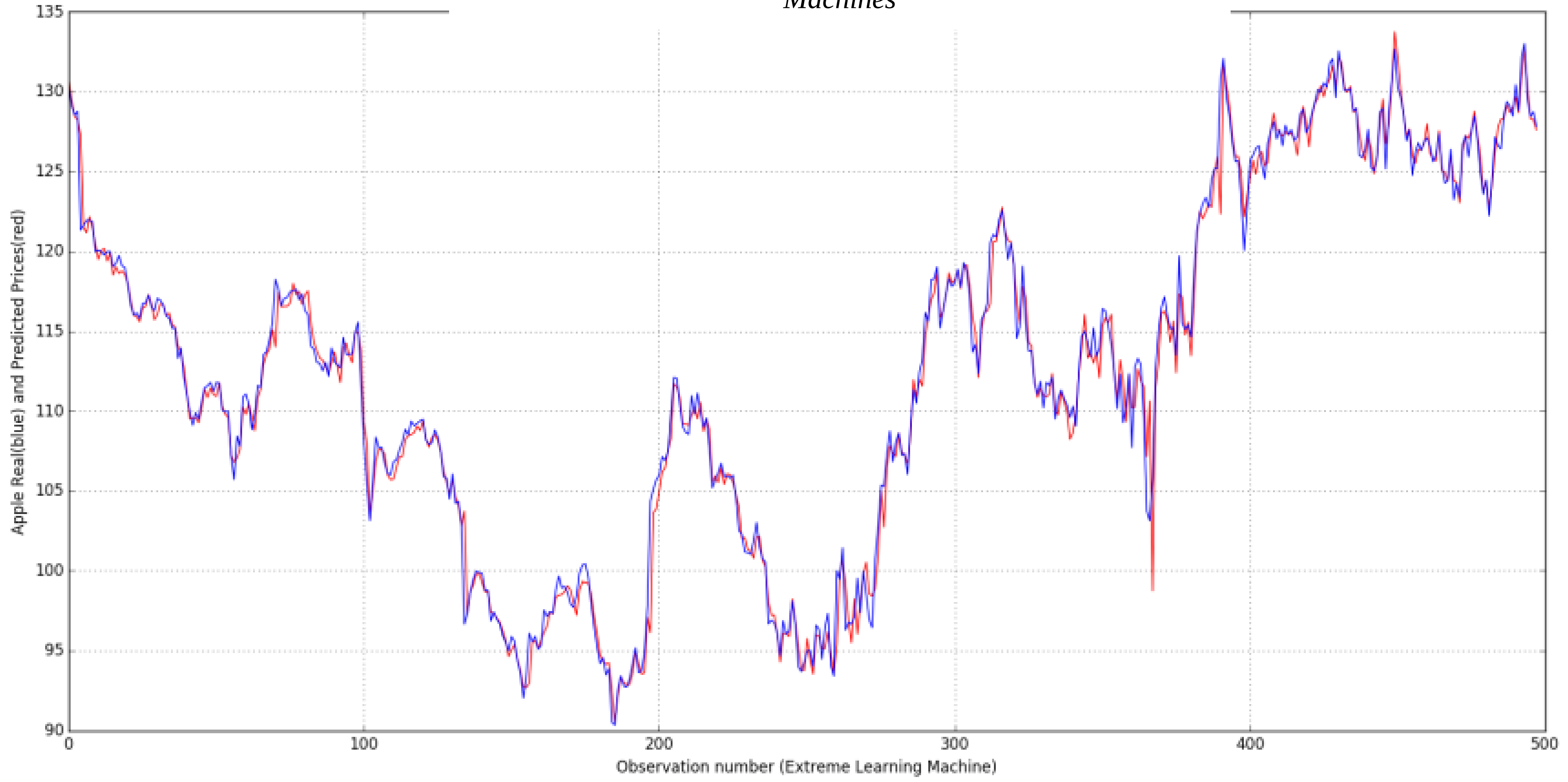


Figure 11.7: Forecasting Bank of America with Extreme Learning Machines

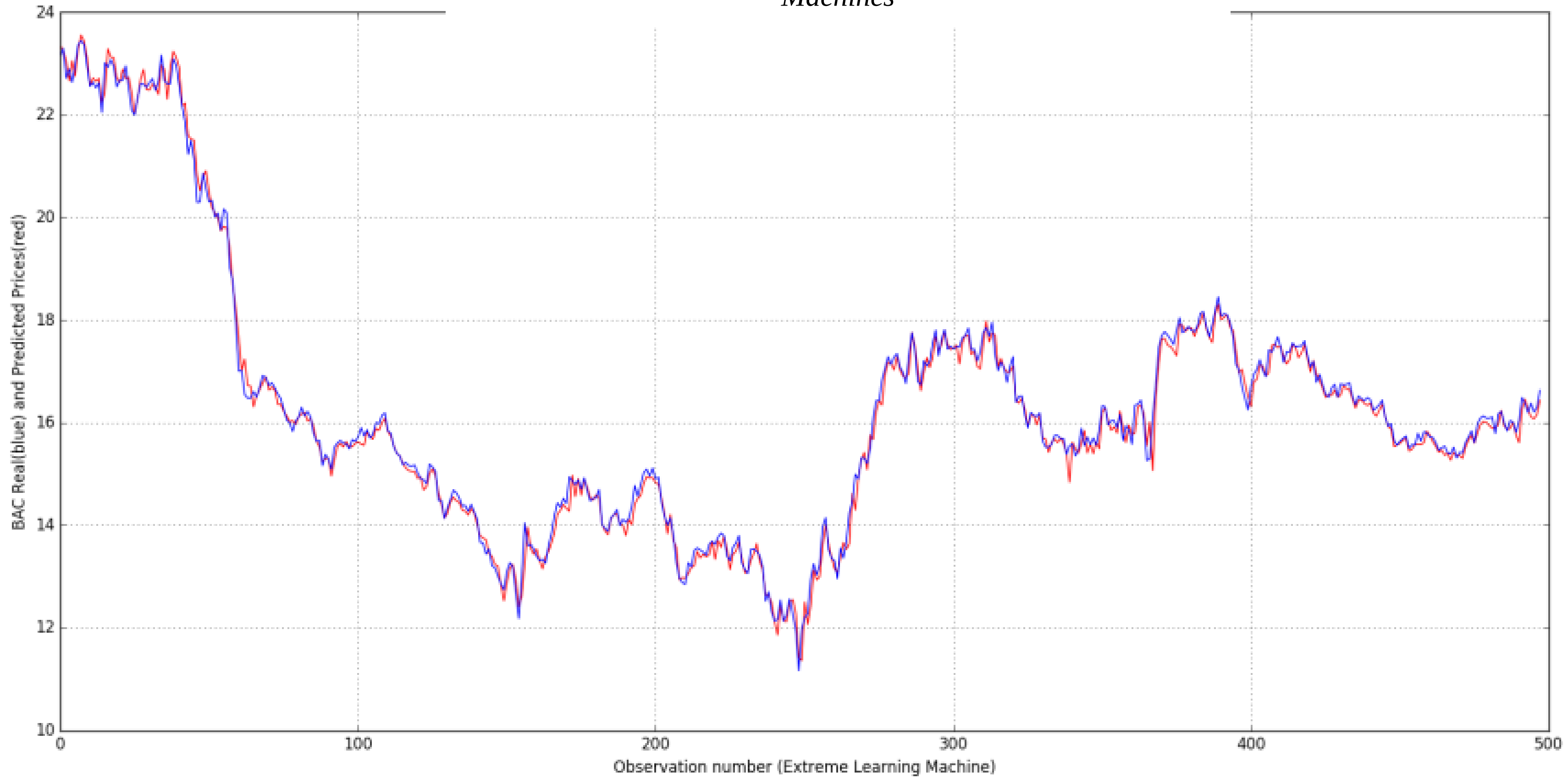
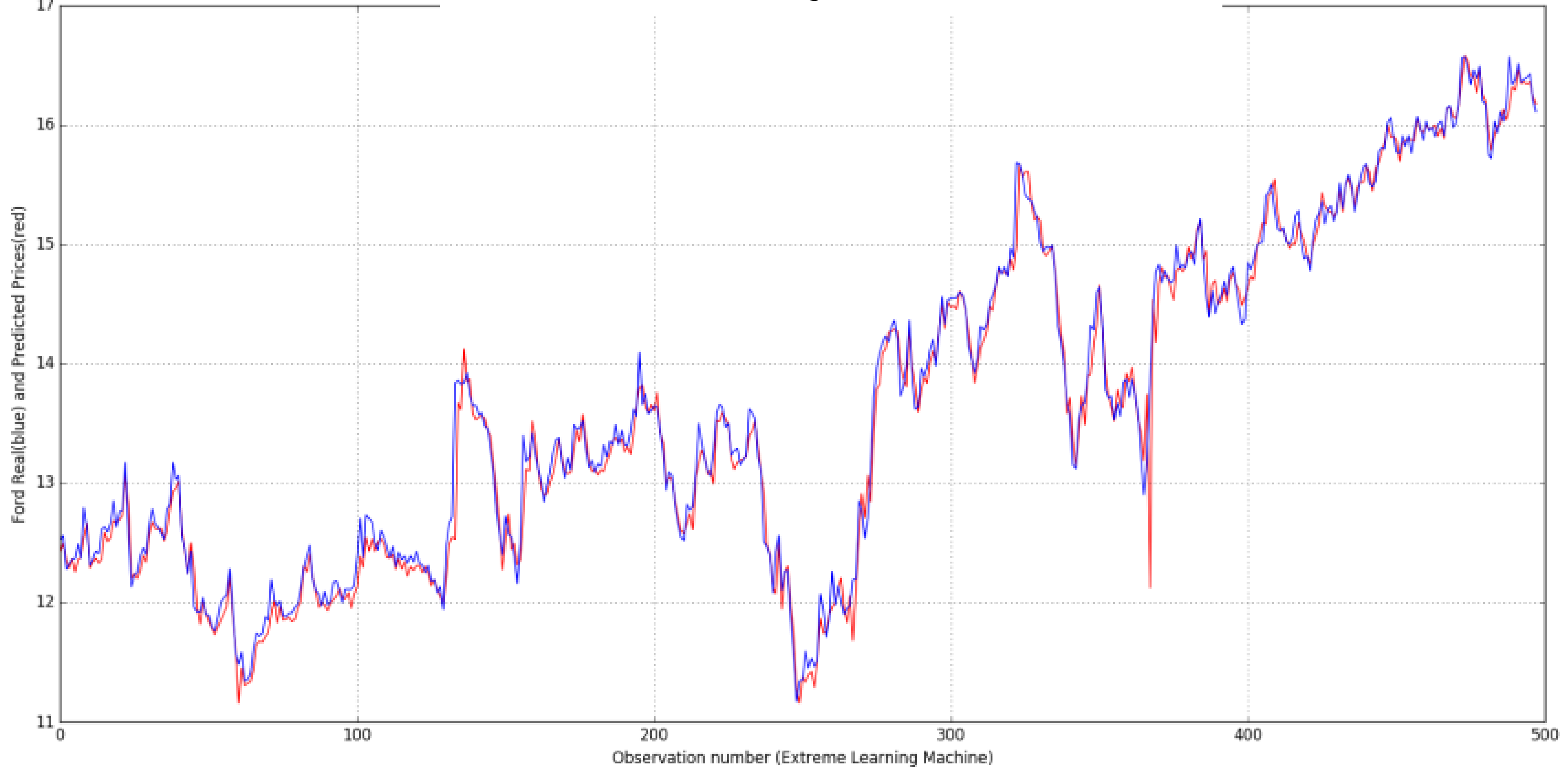


Figure 11.8: Forecasting Ford Motor Company with Extreme Learning Machines



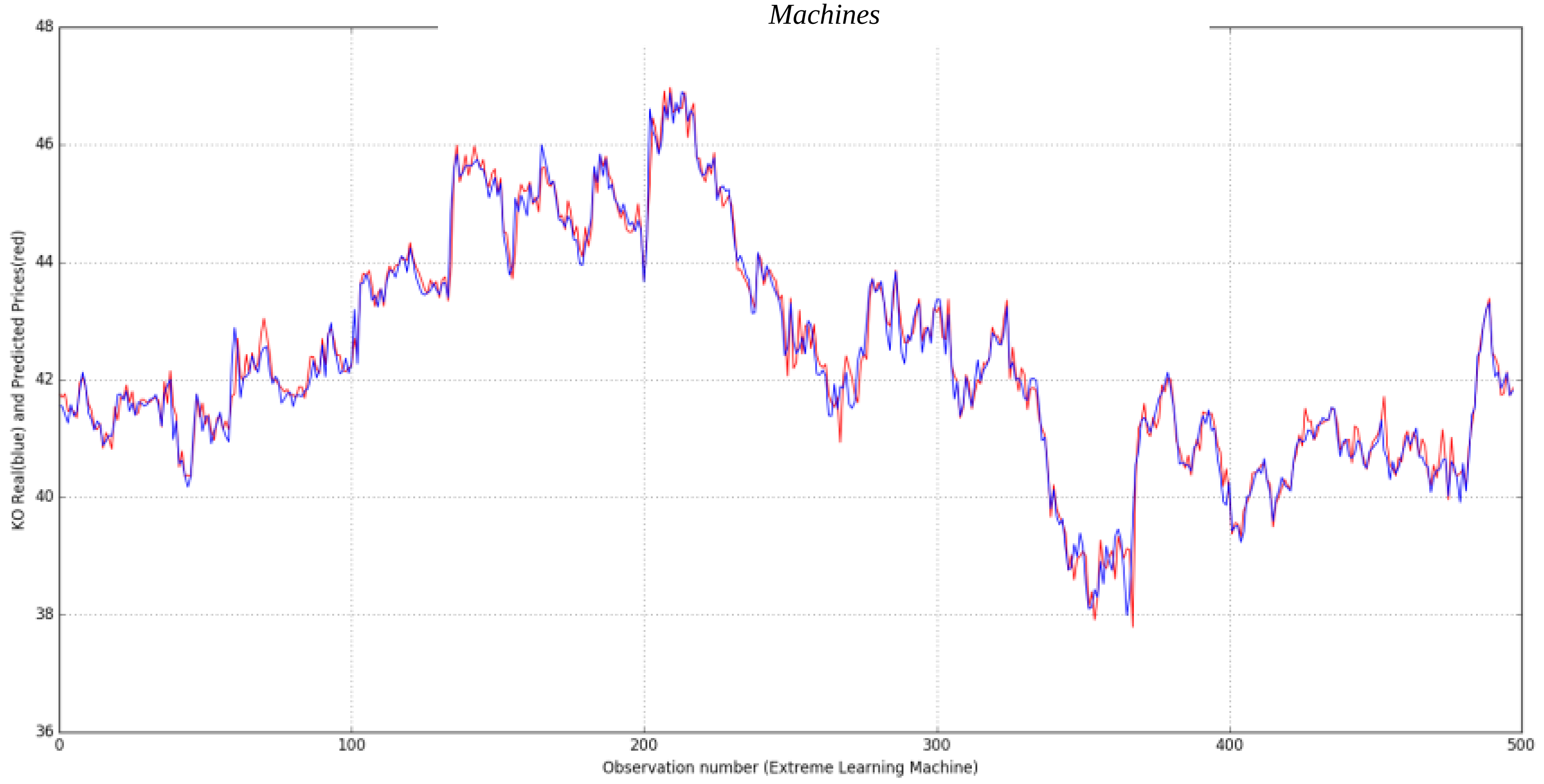
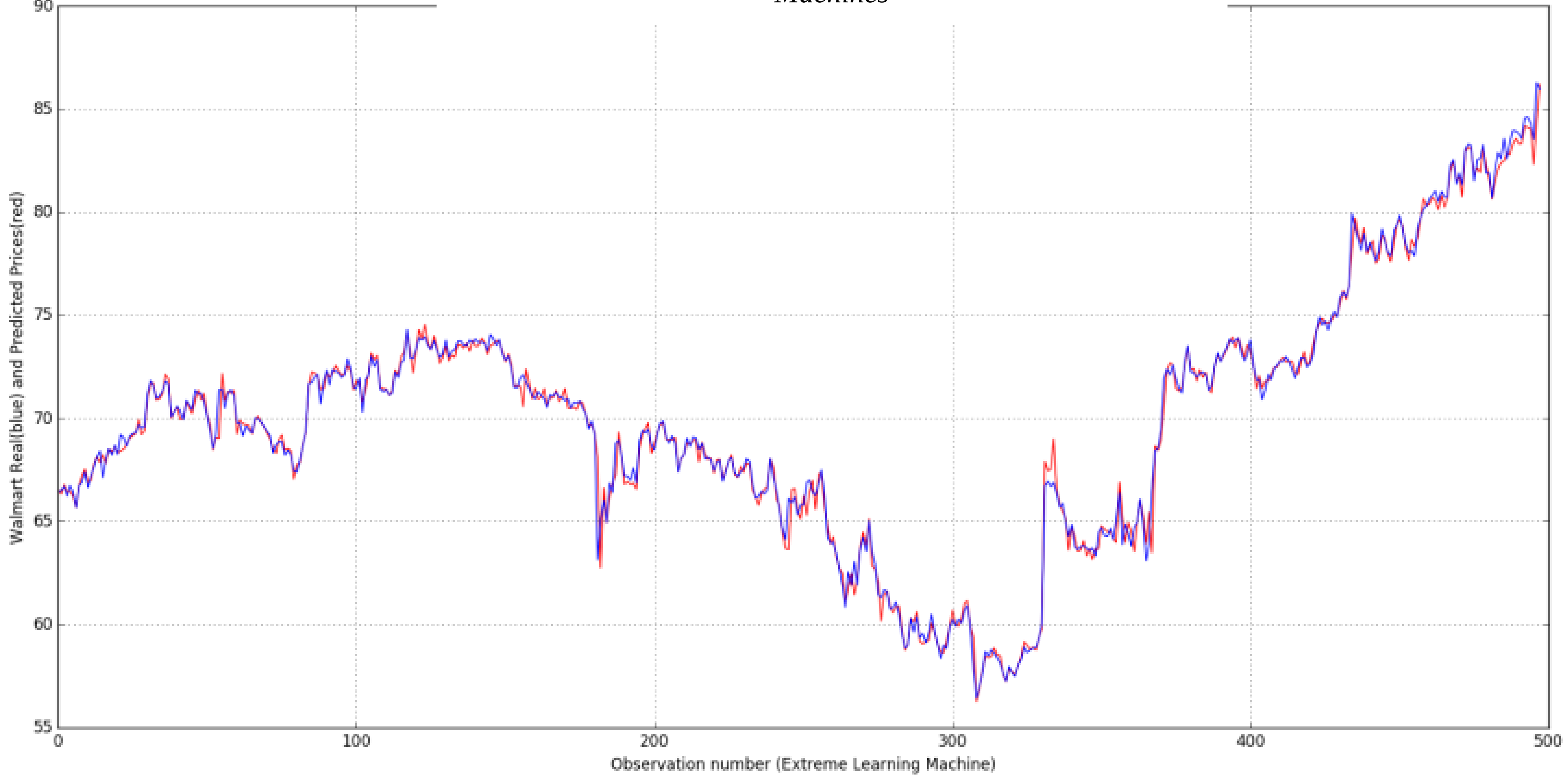


Figure 11.9: Forecasting Coca-Cola with Extreme Learning Machines

Figure 11.10: Forecasting Walmart with Extreme Learning Machines



11.2 Source Code

11.2.1 Extreme Learning Machine Script

```
###################################
#File: elm_model.py
#Author:  Andrew Linzie
#Advisor: Dr. Mirek Mystkowski
#Thesis: Financial Analysis with Artificial Neural Networks
#Date:    April 7, 2017
###################################
# Packages
###################################
import matplotlib.pyplot as plt
from hpelm import HPELM
import numpy as np
import math
import csv
import time
###################################
# Parameters
###################################
i = 5
o = 1
w = 0
testing_csv = "Bac_test.csv"
close_min = 3.14
close_max = 124.0
range_min = -1
range_max = 1
###################################
# Reading in Real Stock Values (.hdf5)
###################################
close_tmp = []

with open(testing_csv, 'rb') as f:
    reader = csv.reader(f)
    reader.next()
    for row in reader:
        close_tmp.append(row[4])      #row[4] = closing values
f.close()
close = map(float, close_tmp)
close_o = np.asarray(close[i + w:(len(close))])

###################################
# Creating the ELM Model
###################################
elm_model = HPELM(i*5, o, '', w=None, batch=1000, accelerator='none',
precision='double', norm=1)
elm_model.add_neurons(1000, 'tanh')

###################################
# Training the ELM model
###################################
start_time = time.time()
```

```
elm_model.add_data("BAC_training_input_i5o1w0_s8299.hdf5","BAC_training_output_i5
o1w0_s8299.hdf5", fHH="HH.hdf5", fHT="HT.hdf5")
elm_model.validation_corr("HH.hdf5", "HT.hdf5",
"BAC_Validate_input_i5o1w0_s498.hdf5","BAC_Validate_output_i5o1w0_s498.hdf5",
steps=100)
end_time = time.time()

##################################
# Testing the ELM model
##################################
y = elm_model.predict("BAC_Testing_input_i5o1w0_s498.hdf5")

#reverse Min Max scaling
close_std = (y - range_min)/(range_max - range_min)
prediction = close_std*(close_max - close_min) + close_min

##################################
# Quantifying Error
##################################
#Mean-Squared Error
mse = elm_model.error(close_o, prediction)

#Type 1 Hit/Miss ratio
count = 0
hit_1 = 0
miss_1 = 0
while(count < len(close_o) - 1):
    if(math.copysign(1, prediction[count] - prediction[count+1]) ==
math.copysign(1, close_o[count] - close_o[count + 1])):
        hit_1 += 1
    else:
        miss_1 += 1
    count += 1

#Type 2 Hit/Miss ratio
count = 0
hit_2 = 0
miss_2 = 0
while(count < len(close_o) - 2):
    if(math.copysign(1, close_o[count] - close_o[count + 1]) != math.copysign(1,
close_o[count + 1] - close_o[count + 2])):
        if(math.copysign(1, close_o[count] - close_o[count + 1]) ==
math.copysign(1, prediction[count] - prediction[count + 1])):
            if(math.copysign(1, close_o[count + 1] - close_o[count + 2]) ==
math.copysign(1, prediction[count + 1] - prediction[count + 2])):
                hit_2 += 1
            else:
                miss_2 += 1
    count += 1

##################################
# Terminal Output
##################################
print "Training time: " +  str(end_time - start_time)
print 'MSE: ' + str(mse)
print "Hit/Miss (type 1): " + str(hit_1) + "/" + str(miss_1)
print "Hit/Miss (type 2):" + str(hit_2) + "/" + str(miss_2)

##################################
# Graphing Real vs Forecasted
##################################
x_axis = np.array(range(len(y)))
plt.plot(x_axis, prediction, 'r')
plt.plot(x_axis, close_o, 'b')
plt.ylabel('Bank of America Company: Real(blue) and Forecasted Prices(red)')
plt.xlabel('Observation number (Extreme Learning Machine)')
plt.grid()
plt.show()
```

11.2.2 Back-Propagation with Stochastic Gradient Descent Script

```
##################################
#File: mlp_stock_regressor.py
#Author:  Andrew Linzie
#Advisor: Dr. Mirek Mystkowski
#Thesis: Financial Analysis with Artificial Neural Networks
#Date:    April 7, 2017
##################################
# Packages
##################################
import numpy as np
import h5py
import csv
import time
import sklearn.metrics as metric
import math
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPRegressor
##################################
# Parameters
##################################
X = []
T = []
i= 5
o= 1
w= 0
testing_csv = "Apple_test.csv"
close_min = 11.0
close_max = 702.100021
range_min = -1
range_max = 1
###########################
# Reading Training Arrays (.hdf5)
###########################
with h5py.File('Apple_training_input_i5o1w0_s8110.hdf5', 'r') as hf:
    #print('List of arrays in this file: \n', hf.keys())
    data = hf.get('data')
    APPL_1X = np.array(data)

with h5py.File('Apple_training_output_i5o1w0_s8110.hdf5', 'r') as hf:
    #print('List of arrays in this file: \n', hf.keys())
    data = hf.get('data')
    APPL_1T = np.array(data)

with h5py.File('Apple_Validate_input_i5o1w0_s498.hdf5', 'r') as hf:
    #print('List of arrays in this file: \n', hf.keys())
    data = hf.get('data')
    APPL_2X = np.array(data)

with h5py.File('Apple_Validate_output_i5o1w0_s498.hdf5', 'r') as hf:
    #print('List of arrays in this file: \n', hf.keys())
    data = hf.get('data')
    APPL_2T = np.array(data)

###########################
# Reading Testing Arrays (.hdf5)
###########################
with h5py.File('Apple_Testing_input_i5o1w0_s498.hdf5', 'r') as hf:
    data = hf.get('data')
    close_i_s = np.array(data)

with h5py.File('Apple_Testing_output_i5o1w0_s498.hdf5', 'r') as hf:
    data = hf.get('data')
    close_o_s = np.array(data)

close_tmp = []
with open(testing_csv, 'rb') as f:
    reader = csv.reader(f)
    reader.next()
    for row in reader:
        close_tmp.append(row[4])        #row[4] = closing values
f.close()
close = map(float, close_tmp)
close_o = np.asarray(close[i + w:(len(close))])

X.extend(APPL_1X)
X.extend(APPL_2X)
T.extend(APPL_1T)
T.extend(APPL_2T)

###########################
# Creating the MLP Model
###########################

reg = MLPRegressor(hidden_layer_sizes=(40),
                   activation='tanh',
                   solver='sgd',
                   alpha=0.001,
                   batch_size=10,
                   learning_rate='constant',
                   learning_rate_init=0.001,
                   power_t=0.5,
                   max_iter=5000,
                   shuffle=True,
                   random_state=None,
                   tol=0.000001,
                   verbose=True,
                   warm_start=False,
                   momentum=0.9,
                   nesterovs_momentum=True,
                   early_stopping=True,
                   validation_fraction=0.05)

###########################
# Training the MLP Model
###########################
start_time = time.time()
reg.fit(X,T)
end_time = time.time()

###########################
# Testing the MLP Model
###########################
y = reg.predict(close_i_s)

#reverse Min Max scaling
close_std = (y - range_min)/(range_max - range_min)
prediction = close_std*(close_max - close_min) + close_min

##################################
# Quantifying Error
##################################
#Mean squared Error
mse = metric.mean_squared_error(close_o, prediction)

#Type 1 Hit/Miss Ratio
count = 0
hit_1 = 0
miss_1 = 0
while(count < len(close_o) - 1):
    if(math.copysign(1, prediction[count] - prediction[count+1]) ==
math.copysign(1, close_o[count] - close_o[count + 1])):
        hit_1 += 1
    else:
        miss_1 += 1
    count += 1

#Type 2 Hit/Miss Ratio
count = 0
hit_2 = 0
miss_2 = 0
while(count < len(close_o) - 2):
    if(math.copysign(1, close_o[count] - close_o[count + 1]) != math.copysign(1,
close_o[count + 1] - close_o[count + 2])):
        if(math.copysign(1, close_o[count] - close_o[count + 1]) ==
math.copysign(1, prediction[count] - prediction[count + 1])):
            if(math.copysign(1, close_o[count + 1] - close_o[count + 2]) ==
math.copysign(1, prediction[count + 1] - prediction[count + 2])):
                hit_2 += 1
            else:
                miss_2 += 1
    count += 1

##################################
# Terminal Output
##################################
print "Training time: " +  str(end_time - start_time)
print 'MSE: ' + str(mse)
print "Hit/Miss (type 1) " + str(hit_1) + "/" + str(miss_1)
print "Hit/Miss (type 2): " + str(hit_2) + "/" + str(miss_2)

##################################
# Graphing Real vs Forecasted
##################################
x_axis = np.array(range(len(y)))
plt.plot(x_axis, prediction, 'r')
plt.plot(x_axis, close_o, 'b')
plt.ylabel('Apple Inc. Real(blue) and Predicted Prices(red)')
plt.xlabel('Observation number (SGD Method)')
plt.grid()
plt.show()
```

11.2.3 Script for Creating Training/Validation Sets

```
##################################
#File: train_set_generator.py
#Author:  Andrew Linzie
#Advisor: Dr. Mirek Mystkowski
#Thesis: Financial Analysis with Artificial Neural Networks
#Date:    April 7, 2017
##################################
# Packages
##################################
import csv
import numpy as np
import hpelm as modules
from sklearn import preprocessing

##################################
# Parameters
##################################
stock_csv = "Apple_Validate.csv"
stock_name = "Apple"
array_purpose = "Validate"
i = 5
o = 1
w = 0

##################################
# Reading in Daily Stock Data
##################################
open_tmp = []
high_tmp = []
low_tmp = []
close_tmp = []
```

```
volume_tmp = []

with open(stock_csv, 'rb') as f:
    reader = csv.reader(f)
    reader.next()
    for row in reader:
        open_tmp.append(row[1])          #row[1] = Open
        high_tmp.append(row[2])          #row[2] = high
        low_tmp.append(row[3])           #row[3] = low
        close_tmp.append(row[4])         #row[4] = close
        volume_tmp.append(row[5])        #row[5] = volume
f.close()

open = map(float, open_tmp)
high = map(float, high_tmp)
low = map(float, low_tmp)
close = map(float, close_tmp)
volume = map(float, volume_tmp)

##################################
# Scaling Training/Validation data
##################################
#Min max Scaling
open_s = preprocessing.minmax_scale(open, feature_range=(-1,1))
high_s = preprocessing.minmax_scale(high, feature_range=(-1,1))
low_s = preprocessing.minmax_scale(low, feature_range=(-1,1))
close_s = preprocessing.minmax_scale(close, feature_range=(-1,1))
volume_s = preprocessing.minmax_scale(volume, feature_range=(-1,1))

#Truncating into training/validation arrays
open_i = open_s[0:(len(open) - o - w)]
open_o = open_s[i + w:(len(open))]
high_i = high_s[0:(len(high) - o - w)]
high_o = high_s[i + w:(len(high))]
low_i = low_s[0:(len(low) - o - w)]
low_o = low_s[i + w:(len(low))]
close_i = close_s[0:(len(close) - o - w)]
close_o = close_s[i + w:(len(close))]
volume_i = volume_s[0:(len(volume) - o - w)]
volume_o = volume_s[i + w:(len(volume))]

##################################
# Formatting Scaled Values into
# Ordered Training Arrays
##################################
#Input
input = []
count = 0
comp_index = 0
while (count < len(open_i) and len(open_i) - count >= i):
    X_i = []
    while (comp_index < i):
        X_i.append(open_i[count+comp_index])
        X_i.append(high_i[count+comp_index])
        X_i.append(low_i[count+comp_index])
        X_i.append(close_i[count+comp_index])
        X_i.append(volume_i[count+comp_index])
        comp_index = comp_index + 1
    input.append(X_i)
    comp_index = 0
    count = count + 1
X = np.array(input)

#Target
output = []
count = 0
comp_index = 0
while (count < len(open_o) and len(open_o) - count >= o):
    T_i = []
    while (comp_index < o):
        T_i.append(close_o[count+comp_index])
        comp_index = comp_index + 1
    output.append(T_i)
    comp_index = 0
    count = count + 1
T = np.array(output)

##################################
# Naming Training/Validation Arrays
##################################
#Input Name
i_name = stock_name + '_' + array_purpose + '_input_i' + str(i) + 'o' + str(o) +
'w' + str(w) + '_s' + str(len(input)) + '.hdf5'

#Target Name
o_name = stock_name + '_' + array_purpose + '_output_i' + str(i) + 'o' + str(o) +
'w' + str(w) + '_s' + str(len(output)) + '.hdf5'

##################################
# Saving to .hdf5 File Format
##################################
try:
    mode = int(raw_input('Would you like to save these HDF5 files to disk?
(yes=1, no=0): '))
except ValueError:
    print 'Not a number'
if (mode == 1):
    modules.make_hdf5(X, i_name)
    modules.make_hdf5(T, o_name)
    print 'The files were written to the disk.'
else:
    print 'The files were not written to the disk.'
```

11.2.4 Script for Creating Testing Sets

```
##################################
#File: test_set_generator.py
#Author:  Andrew Linzie
#Advisor: Dr. Mirek Mystkowski
#Thesis: Financial Analysis with Artificial Neural Networks
```

```
#Date:    April 7, 2017
##################################
# Packages
##################################
import hpelm as modules
import numpy as np
import csv

##################################
# Parameters
##################################
historical_csv = 'Apple_Training.csv'
stock_csv = 'Apple_test.csv'
array_purpose = "Testing"
stock_name = "Apple"
i = 5
o = 1
w = 0

##################################
# Reading in Daily Stock Data
##################################
open_tmp = []
high_tmp = []
low_tmp = []
close_tmp = []
volume_tmp = []

open_recent = []
high_recent = []
low_recent = []
close_recent = []
volume_recent = []

with open(stock_csv, 'rb') as f:
    reader = csv.reader(f)
    reader.next()
    for row in reader:
        open_tmp.append(row[1])          #row[1] = Open
        high_tmp.append(row[2])          #row[2] = high
        low_tmp.append(row[3])           #row[3] = low
        close_tmp.append(row[4])         #row[4] = close
        volume_tmp.append(row[5])        #row[5] = volume
f.close()

with open(historical_csv, 'rb') as g:
    reader = csv.reader(g)
    reader.next()
    for row in reader:
        open_recent.append(row[1])          #row[1] = Open
        high_recent.append(row[2])          #row[2] = high
        low_recent.append(row[3])           #row[3] = low
        close_recent.append(row[4])         #row[4] = close
        volume_recent.append(row[5])        #row[5] = volume
g.close()
```

```
open = map(float, open_tmp)
high = map(float, high_tmp)
low = map(float, low_tmp)
close = map(float, close_tmp)
volume = map(float, volume_tmp)

e_open = map(float, open_recent)
e_high = map(float, high_recent)
e_low = map(float, low_recent)
e_close = map(float, close_recent)
e_volume = map(float, volume_recent)

##################################
# Scaling Testing data
##################################
#Getting Min/Max for each feature
open_min = np.amin(e_open)
open_max = np.amax(e_open)
high_min = np.amin(e_high)
high_max = np.amax(e_high)
low_min = np.amin(e_low)
low_max = np.amax(e_low)
close_min = np.amin(e_close)
close_max = np.amax(e_close)
volume_min = np.amin(e_volume)
volume_max = np.amax(e_volume)

#Need to know close_min and close_max for reverse scaling
print "Close_min: " + str(close_min) + ", Close_max: " + str(close_max)

#Min max Scaling
open_std = (open - open_min) / (open_max - open_min)
open_s = open_std * (1 - (-1)) + (-1)
high_std = (high - high_min) / (high_max - high_min)
high_s = high_std * (1 - (-1)) + (-1)
low_std = (low - low_min) / (low_max - low_min)
low_s = low_std * (1 - (-1)) + (-1)
close_std = (close - close_min) / (close_max - close_min)
close_s = close_std * (1 - (-1)) + (-1)
volume_std = (volume - volume_min) / (volume_max - volume_min)
volume_s = volume_std * (1 - (-1)) + (-1)

#Truncating into testing arrays
open_i = open_s[0:(len(open) - o - w)]
open_o = open_s[i + w:(len(open))]
high_i = high_s[0:(len(high) - o - w)]
high_o = high_s[i + w:(len(high))]
low_i = low_s[0:(len(low) - o - w)]
low_o = low_s[i + w:(len(low))]
close_i = close_s[0:(len(close) - o - w)]
close_o = close_s[i + w:(len(close))]
volume_i = volume_s[0:(len(volume) - o - w)]
volume_o = volume_s[i + w:(len(volume))]
```

```
##################################
# Formatting Scaled Values into
# Ordered Testing Arrays
##################################
#Input
input = []
count = 0
comp_index = 0
while (count < len(open_i) and len(open_i) - count >= i):
    X_i = []
    while (comp_index < i):
        X_i.append(open_i[count+comp_index])
        X_i.append(high_i[count+comp_index])
        X_i.append(low_i[count+comp_index])
        X_i.append(close_i[count+comp_index])
        X_i.append(volume_i[count+comp_index])
        comp_index = comp_index + 1
    input.append(X_i)
    comp_index = 0
    count = count + 1
X = np.array(input)

#Target
output = []
count = 0
comp_index = 0
while (count < len(open_o) and len(open_o) - count >= o):
    T_i = []
    while (comp_index < o):
        T_i.append(close_o[count+comp_index])
        comp_index = comp_index + 1
    output.append(T_i)
    comp_index = 0
    count = count + 1
T = np.array(output)

##################################
# Naming Testing Arrays
##################################
##Input Name
i_name = stock_name + '_' + array_purpose + '_input_i' + str(i) + 'o' + str(o) +
'w' + str(w) + '_s' + str(len(input)) + '.hdf5'

#Target Name
o_name = stock_name + '_' + array_purpose + '_output_i' + str(i) + 'o' + str(o) +
'w' + str(w) + '_s' + str(len(output)) + '.hdf5'

##################################
# Saving to .hdf5 File Format
##################################
try:
    mode = int(raw_input('Would you like to save these HDF5 files to disk?
(yes=1, no=0): '))
except ValueError:
    print 'Not a number'
```

```
if (mode == 1):
    modules.make_hdf5(X, i_name)
    modules.make_hdf5(T, o_name)
    print 'The files were written to the disk.'
else:
    print 'The files were not written to the disk.'
```